Document DTI-UM-SP101-3
Price $25.00

User Manual

for

SP-101 V2

DTLIB Real-Time Peripheral Support

under

FORTRAN/RT-11

For use with RT-11 V4 and FORTRAN/RT-11 V2.5

SP-101 V02-01

Data Translation, Inc.
100 Locke Drive
Marlboro, Massachusetts 01752

Second Edition
September 1980

Preface

This manual describes the usage and configuration of DTLIB, Data Translation's library of real-time peripheral support routines.

The manual is broken up into the following chapters:

Chapter 1 **Using DTLIB**

This chapter describes the procedures required to use DTLIB with FORTRAN/RT-11 and the system requirements and constraints.

Chapter 2 **Performance Considerations and Applications**

This chapter discusses the system performance and throughput considerations to be kept in mind by the user to optimize program performance. Some applications are discussed and sample programs presented.

Chapter 3 **Descriptions of DTLIB Routines and Functions**

This chapter describes the parameter lists and calling syntax of all of the subroutines and functions in DTLIB.

Chapter 4 **System Integration**

This chapter covers the procedures to correctly configure DTLIB for a given hardware arrangement. It is important that the user read this chapter carefully to fully understand how to configure the hardware system and to have DTLIB operate correctly with the given hardware configuration.

DTLIB Real-Time Peripheral Support

User Manual

CONTENTS

CONTENTS

(continued)

CHAPTER 1

Using DTLIB

1.1 Introduction

DTLIB is a real-time support package for Data Translation's wide offering of laboratory and industrial peripherals. Consisting of a single library of subroutines that are called from user FORTRAN programs, DTLIB allows the laboratory or industrial user to initiate and control operations on the following types of hardware devices:

Model No.	Hardware Description
DT2762	an analog-to-digital converter with 12,14,or 16 bit resolution and optional programmable gain amplifier. Accepts 16 Single Ended or 8 Differential input channels (up to 64 SE or 32 DI input channels with DT2772 expander). Performance-enhanced version of DEC ADV-11 option.
DT2764	a wide-range analog-to-digital converter with 12,14,or 16 bit resolution and optional programmable gain amplifier, allowing full scale input ranges from 10 mV to 10 V. Accepts 16 Single Ended or 8 Differential input channels (up to 64 SE or 32 DI input channels with DT2774 expander).
DT2765	an isolated-input analog-to-digital converter with 12 bit resolution and optional programmable gain amplifier. Accepts 4 Differential input channels (up to 60 DI input channels with multiple DT2775 expanders). Withstands up to 250 V common-mode voltage on inputs.
DT2766	a four channel digital-to-analog converter with 4 digital output bits for plotter/CRT control. 12 bit resolution. Performance-enhanced version of DEC AAV-11 option.

DT2767	identical to DT2766 except 8 bit resolution.
DT2768	a general purpose 16-bit digital input/output interface. Software and hardware compatible to DEC DRV-11 option.
DT2768-I	Identical to DT2768 except that all input/output lines are optically isolated.
DT2769	a programmable clock/counter combination that determines time intervals or counts events. Software and hardware compatible to DEC KWV-11 option.
DT2771	a high-speed Direct Memory Access digital-to-analog converter designed for CRT graphics applications. Captures "light pen" events. Two output channels and full Z-axis control.
DT2781	a combination analog input/analog output interface system. Contains an analog-to-digital converter with 12 bit resolution accepting 16 Single Ended or 8 Differential input channels. Also contains two digital-to-analog output channels with 12 bit resolution.
DT2782	a high-speed Direct Memory Access analog-to-digital converter with 12, 14, or 16 bit resolution and optional programmable gain amplifier. Accepts 16 Single Ended or 8 Differential input channels.
DT2784	a Direct Memory Access wide range analog-to-digital converter with 12, 14, or 16 bit resolution and optional programmable gain amplifier, allowing full scale input ranges from 10 mV to 10 V. Accepts 16 Single Ended or 8 Differential input channels.
DT2785	a combination analog input/analog output interface system. Contains a wide-range analog-to-digital converter with 12 bit resolution allowing full scale input ranges from 10 mV to 10 V. Accepts 16 Single Ended or 8 Differential input channels. Also contains two digital-to-analog output channels with 12 bit resolution.

Table 1-1 Hardware Descriptions

In addition to providing easy control of peripheral devices, DTLIB also provides general purpose routines to perform such functions as

* convert INTEGER*2 to Binary Coded Decimal (BCD)
* convert Binary Coded Decimal (BCD) to INTEGER*2
* convert REAL*4 to 16 bit straight binary (0-65535. range)
* convert 16 bit straight binary (0-65535. range) to REAL*4
* wait for real-time operation complete

1.2 Using DTLIB calls in FORTRAN programs

It is very simple to use the facilities provided by DTLIB inside user FORTRAN programs. All that is necessary to call a DTLIB routine is to include a statement such as

```
CALL     name (argument list)
         or
VARIABLE = name (argument list)
```

Not all DTLIB routines can be called by both of the above forms - see Chapter 3 for the exact calling sequences allowed for each DTLIB routine.

If a call to a DTLIB routine is used as an argument to another FORTRAN routine, the name of the DTLIB routine may have to be declared EXTERNAL to prevent the FORTRAN compiler from reporting an error. The DEC FORTRAN documentation discusses this requirement in the section of the FORTRAN language reference manual defining the EXTERNAL statement. Other than this requirement, it is not necessary to declare DTLIB names as EXTERNAL names.

Each DTLIB routine expects to receive a series of arguments when called. These arguments are passed in a list enclosed by parenthesis immediately following the name of the DTLIB routine. For example,

```
CALL     IADC (argument list here)
```

If more than one argument is required, the arguments must be separated by commas.

In some cases, certain arguments may be left blank (defaulted). In these cases, the argument would be left out of the list entirely. However, the commas both before and after the blank argument must be left in. For example,

```
argument present:      ---,IUNIT,---
argument blank:        ---,,---
```

Chapter 3 contains a complete description of all arguments expected by each DTLIB routine and describes which arguments may be left blank.

1.3 Compiling the FORTRAN program

Once the program text has been prepared with the text editor, the program may be compiled. For information about how to operate the FORTRAN compiler on the user program, consult the DEC RT-11 documentation or the DEC FORTRAN/RT-11 User's

Guide. Note that it is extremely important that all DEC mandatory patches and corrections to the compiler and to the FORTRAN Object Time System (OTS) be implemented for proper operation.

Compile the FORTRAN program containing the DTLIB calls exactly as any other FORTRAN program might be compiled. No special action is necessary. It is, however, strongly recommended that the /NOSWAP option be specified to insure that the RT-11 User Service Routines (USR) are not swapped into and out of memory over the user's FORTRAN code while real-time operations are active.

NOTE

In some cases, the flag variables passed to DTLIB routines (described as ICMF and IBEF in Chapter 3) will be optimized out of expressions and calls by the FORTRAN compiler. This is done transparently with respect to the user. To prevent the compiler from performing optimizations on these flag variables, execute a statement such as

EQUIVALENCE (<DTLIB flag variable>, <dummy name>)

This will insure that the flag variable named in the statement will not be affected by compiler optimization.

1.3.1 Compile-time Errors

The only errors involving DTLIB routines that will be reported by the FORTRAN compiler are those concerning the basic syntax of the CALL statement or the usage of a DTLIB routine name as a function. Typically, these errors would be caused by such typographical errors as leaving out a parenthesis or typing a period instead of a comma. Failure to declare the name of a completion routine used in a DTLIB call as an EXTERNAL name will also generate an error.

1.4 Linking the FORTRAN program

Once the components of the final application program have been separately compiled or assembled, the code modules must be linked together to generate a copy of the program that RT-11 can execute. Consult the DEC RT-11 documentation and the

FORTRAN/RT-11 User's Guide for detailed information on how to run the Linker.

All that is necessary to have the Linker supply the code for the DTLIB routines called from a FORTRAN program is to include the code module named "DTLIB.OBJ" in the list of code modules passed to the Linker. It is not necessary to inform the Linker that this code module (DTLIB.OBJ) is a library of modules; the Linker will detect this by itself and process the library properly. The position of DTLIB.OBJ in the list of code modules to be linked is arbitrary - it can appear anywhere as long as it is included in the list.

The Linker will only insert code from DTLIB for the routines that are actually used - no extra code will be added that will not be accessed. Only those routines referenced in the FORTRAN program will be inserted in the final run-copy. Note that both the FORTRAN Object Time System (OTS) and the RT-11 FORTRAN IV system library must be properly installed.

1.4.1 Link-time Errors

The only errors that might be generated by DTLIB at link-time would occur if the particular DTLIB.OBJ library that was being used did not contain all of the DTLIB routines that were called by the FORTRAN program. This might occur if the user deleted certain routines from the library module (not recommended practice) and then referenced the deleted routines.

1.5 Executing the FORTRAN Program

Information about executing the final copy of the FORTRAN program will be found in the DEC RT-11 documentation and the FORTRAN/RT-11 User's Guide. FORTRAN-generated run-time errors will also be documented in those references.

NOTE

The RT-11 SJ monitor will not make any attempt when a user FORTRAN program exits to shut down active interrupt-driven I/O. It is strongly recommended that the user type the monitor RESET command after a program exit to insure that all real-time sampling has stopped before another program is loaded into memory. The FB monitor uses a DTLIB-supplied table to explicitly stop all DTLIB peripherals.

1.5.1 Run-time Errors

DTLIB performs all of its error-checking at run-time. Syntax, argument values, hardware presence, and hardware usage are all checked and any errors reported to the user. Most DTLIB-detected errors will generate a printed message similar to

?DTLIB- <error message here>

followed by a FORTRAN error code 0 message. The FORTRAN Object Time System (OTS) will print the line number of the source code containing the error.

In addition, if the DTLIB completion-routine scheduler encounters a major scheduling problem (such as trying to queue more than 32767 executions of a completion routine at once), a message such as

?SYSLIB-F-Interrupt overrun

will be printed followed by a FORTRAN error code 0 message.

NOTE

Leaving an argument blank in a position where DTLIB expects an argument may generate a FORTRAN 61 error when certain DTLIB routines attempt to reference the mandatory argument. This problem is generally caused by a typographical mistake such as leaving out commas in a DTLIB call where other arguments are defaulted. This error may also be generated if the DTLIB object library in use does not contain the correct information about the hardware resident in the system.

1.5.2 Summary of DTLIB errors

vector protection conflict (FORTRAN error code 0 and DTLIB message)

The interrupt vector of one or more of the peripherals that DTLIB expects to control is already in use by the other job under the FB monitor. Only one job can control a single peripheral at a time.

wrong number of arguments (FORTRAN error code 16)

Too many or too few arguments were passed to a DTLIB

routine.

syntax (FORTRAN error code 0 and DTLIB message)

There is some syntax error in the DTLIB call (such as defaulting a mandatory argument, or not specifying the minimum number of required arguments for a given mode).

argument value (FORTRAN error code 0 and DTLIB message)

One (or more) of the arguments passed to the DTLIB routine have values that are illegal. Check the description of the routine in Chapter 3 against the offending source program line carefully to determine which argument is in error.

buffer argument value (FORTRAN error code 0 and DTLIB message)

In the DTLIB routines that collect large quantities of data (RTS, ISORTS, HIST, DRS), this error indicates that one of the following conditions has occurred during set-up:

1. ISIZ is negative or zero
2. The address of IBUF + ISIZ is too large (memory wrap-around)
3. NBUF is negative or zero
4. NREAD is zero
5. NBUF is larger than ISIZ

Check the description of the routine in Chapter 3 against the offending source program line carefully to determine which argument is in error.

sub-buffer argument value (FORTRAN error code 0 and DTLIB message)

In the DTLIB routines that collect large quantities of data using more then one sub-buffer, this error indicates that one of the following conditions has occurred during set-up:

1. the user-specified sub-buffers will not hold an integral number of data points. Remember

that under certain conditions many of the routines require more than one INTEGER*2 location to store a data point.

2. NREAD does not represent an integral number of sub-buffers. NREAD must be a multiple of the number of data points that will fill a sub-buffer. Only complete sub-buffers are collected - no partial sub-buffers are allowed.

3. the number of INTEGER*2 locations per data-point is zero (e.g. NCCH=0 in an RTS call).

Check the description of the routine in Chapter 3 against the offending source program line carefully to determine which argument is in error.

device not present (FORTRAN error code 0 and DTLIB message)

A DTLIB routine has been called that references a hardware interface that DTLIB does not believe exists in the user system. Check the source program to see if the correct DTLIB call is being made, or check the DTLIB configuration module to insure that all peripherals to be controlled are included.

illegal hardware/software usage
(FORTRAN error code 0 and DTLIB message)

A DTLIB routine has been called in a particular mode of operation that is illegal. In most cases, this indicates that a routine is being called in a new set-up mode while there are still completion routines queued for execution from a previous collection operation. If completion routines are active, the DTLIB routine that queued the completion routines can only be called in the special "MODIFY" mode.

In addition to the above errors, DTLIB will report an error when a peripheral device under DTLIB control is referenced simultaneously by an asynchronous sampling routine and a program-driven sampling routine (e.g. calling IADC while RTS is active). This type of message will look like

?DTLIB-F-IADC: A/D in use

followed by a FORTRAN error code 0 message.

1.6 Reporting Software Defects

Should the user encounter a software defect in DTLIB while under warranty, Data Translation will correct the defect at no charge to the user. However, in order for this defect to be reported and corrected, a Software Performance Report form must be completely filled out and returned to Data Translation. Copies of this form are available from Data Translation. Source distribution kits are not covered under this warranty.

NOTE

It is extremely important that some other potential sources of errors be checked first before contacting DTI. In particular, the DEC FORTRAN compiler and OTS must be up to the current revision level with all DEC patches. In addition, the user should try compiling the suspect FORTRAN program with the /NOSWAP option or with RT-11 set with the USR in NOSWAP mode before reporting the problem to DTI. FORTRAN's choice of where to swap the USR code can create serious problems in certain real-time programs. See the RT-11 V4 Software Support manual (documentation volume SSM, numbered 3B) for more information on FORTRAN and the USR.

CHAPTER 2

Performance Considerations and Applications

2.1 Introduction

This chapter covers several topics. Starting with a discussion of system data throughputs, the chapter continues to cover the use of completion routines, data collection techniques, and finishes with the presentation of several sample programs (source copies provided in machine readable form with the binary distribution kits).

2.2 System Throughputs

Attempting to determine maximum data throughput rates for DTLIB routines is a very elusive task. There are many interacting factors which affect the maximum throughput rate.

To begin with, operating DTLIB under the Single Job (SJ) RT-11 monitor will allow significantly higher data collection rates than if the Foreground/Background (FB) monitor is used. This is because of a large reduction of system software overhead in the SJ monitor as compared to the FB monitor. Unless other conditions force the use of the FB monitor, the user is urged to use the SJ monitor when collecting data.

System interrupt traffic will also affect data collection rates. If any I/O activity is going on at all (disk transfers, character I/O to the terminal, line time clock operations), high speed data rates will suffer. The degree to which the rates must be decreased to eliminate data overrun errors will vary according to the interrupt traffic being handled.

Users with older LSI-11 processors (quad height) that must utilize either DMA refresh or microcode memory refresh will see a marked performance degradation: every 2 mS the entire bus becomes idle while a burst memory refresh occurs. Data collection cannot occur at all during the refresh intervals -

thus, the maximum continuous rate must be able to handle these bursts being inserted at any time in the sampling process. The DMA refresh uses a distributed refresh arrangement - this technique lessens the impact of refresh on data collection, but will still lower the maximum rates.

Although the use of completion routines (see later section) can improve the overall performance of DTLIB in supporting complex data collection structures, completion routines add system overhead and thus may decrease the hardware maximum data rate.

In any case, if the user's application requires data collection rates that approach the maximum that DTLIB can support, the user is urged to start testing his software at the lowest acceptable rate and increase the rate in small steps until the desired performance is achieved or DTLIB can no longer support the rate.

Appendix B contains tables that represent the maximum data collection rates (throughput) achieved with several DTLIB routines under actual operating conditions. The test conditions and machine configurations are also given in Appendix B. No completion routines were being used, and there was no I/O traffic at the time of the tests other than that related to the test itself. The rates listed are the maximum rates achieved before a data overrun condition was reported by DTLIB. Again, the numbers in Appendix B will probably be different on different systems. The rates are presented so that the user can approximate what to expect.

Note that the primary speed limitation in processing interrupt-driven sampling is software overhead: increases in hardware speed have only a little impact on throughput. This is because of the large maximum interrupt latencies of the LSI-11 (up to 44 uS) along with the minimum operations required by the interrupt handlers. Users with LSI-11/23 processors will see a marked speed improvement due to both instruction execution speed and reduced interrupt latency. There is a "FAST SWEEP" mode in the RTS routine where interrupts are not used, but rather the processor remains in an extremely tight code loop collecting data and doing nothing else. Maximum data rates in this mode are 2.5 to 3 times faster than the interrupt-driven mode, but at the expense of the processor doing nothing but collecting data. The highest data rate in this mode is achieve by locking out all interrupts during the collection interval.

2.3 Buffer Partioning

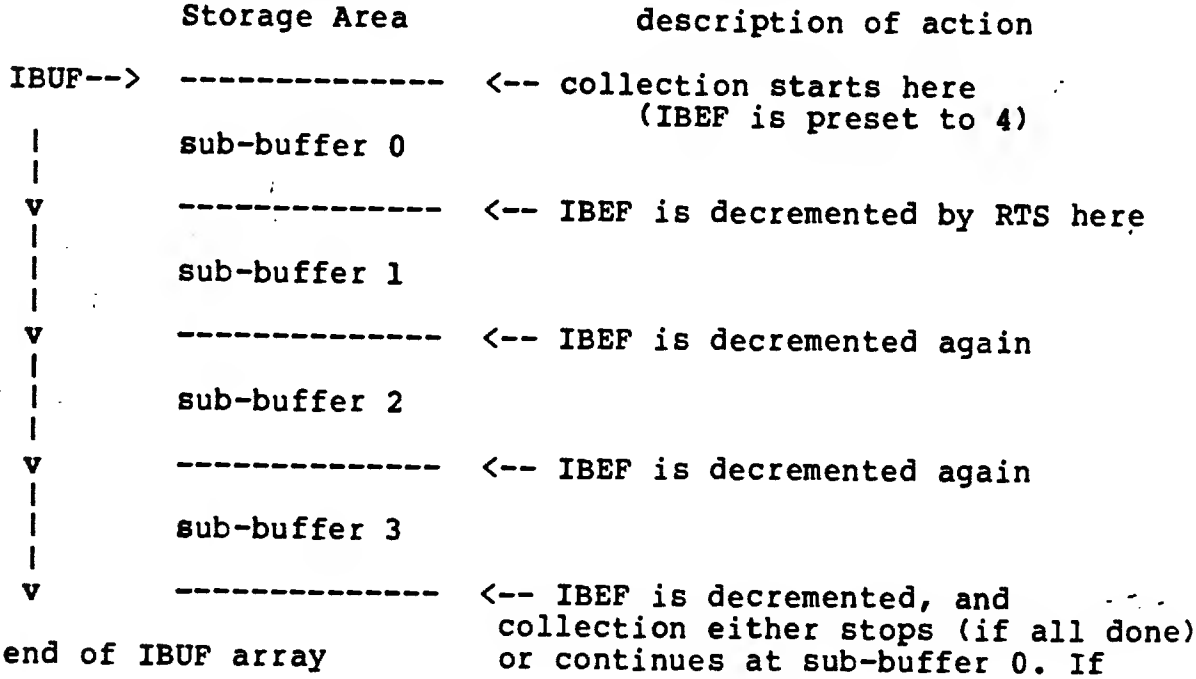
DTLIB offers a powerful data buffer partitioning scheme for use where data is to be separated into groups of data,

where data is to be collected continuously, and where more data is to be collected than will fit in memory at one time.

The key to this scheme lies in the arrangement of the data buffer (IBUF array) used by the asynchronous sampling routines. This buffer may be partitioned into N sections or sub-buffers. Each sub-buffer holds an integral number of data points (note that some routines collect data points that are more than one word long). These N sub-buffers are all the same size, with any excess words in the main buffer being ignored. For example, if a buffer is 235 words, and 10 sub-buffers are requested, each buffer would be 23 words long, with the last 5 words in the main buffer being ignored. DTLIB will report an error if the number of subbuffers requested is larger than the original main buffer, if each sub-buffer will not hold an integral number of data points, or if DTLIB cannot partition the main buffer for the requested number of sub-buffers.

Whenever any one of these sub-buffers is filled, a buffer event flag is decremented automatically. This flag (described as IBEF in Chapter 3) is preset by DTLIB at the start of sampling to the number of sub-buffers available. Thus, at any point, this flag indicates how many sub-buffers are left to fill. It is the use of this flag in conjunction with completion routines or special main program code that allows DTLIB to support continuous sampling.

The following diagram illustrates a main buffer partioned into 4 sub-buffers (NBUF=4), and indicates what occurs as the buffer is filled with data. IBEF is decremented as soon as the contents of the previous sub-buffer is valid. In addition, any completion routine scheduling that is requested is performed at the same time (meaning at the end of each sub-buffer).



collection continues, the user program or completion routine must insure that IBEF never reaches 0 (indicating all sub-buffers full), or DTLIB will report a DATA OVERRUN.

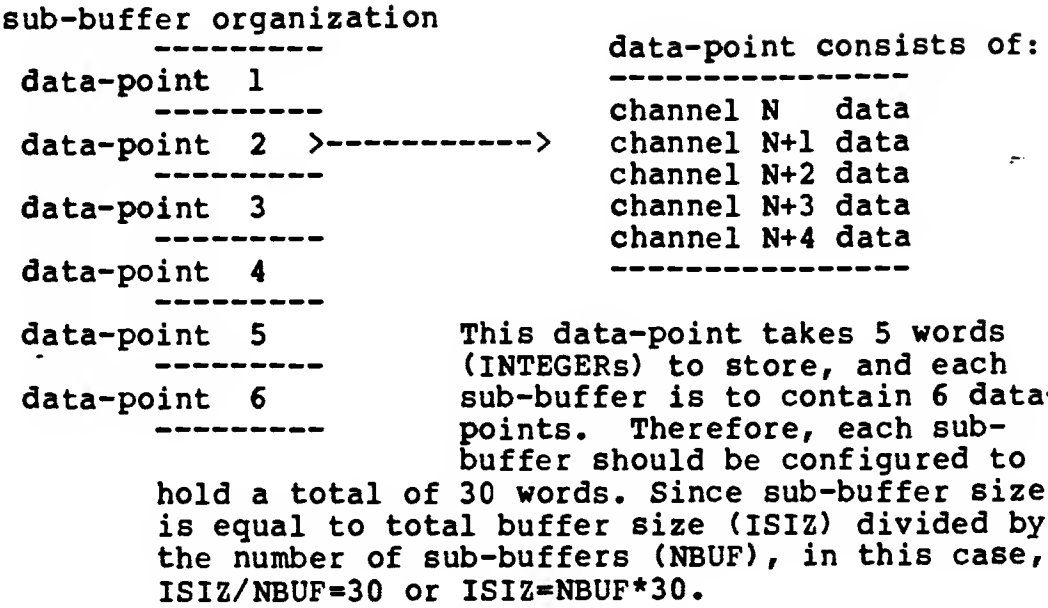
To sample data continuously, the user specifies a negative number of points to the particular DTLIB sampling routine (NREAD < 0). This enables the continuous acquisition mode of DTLIB. Hereafter, DTLIB will collect data continuously until the buffer event flag becomes zero. Ordinarily, the user also arranges for a completion routine to be executed every sub-buffer to process the sub-buffer of data. When the completion routine finishes execution, it increments the buffer event flag, thus informing DTLIB that another sub-buffer is available for data. As long as the average rate of completion routine execution is equal to or greater than the average rate of sub-buffer collection, data can be collected forever. Should the buffer event flag ever reach zero, though, DTLIB will stop and report an overrun error, as this condition indicates that all sub-buffers are full and no more space is available for DTLIB to put data; the completion routine is not emptying sub-buffers as fast as DTLIB is filling them. To stop this continuous sampling, the routine is called in a special stop mode (called "MODIFY MODE").

One note about this continuous mode of collection: DTLIB operates in a ring-buffer fashion while filling sub-buffers. When sub-buffer N-1 is full, DTLIB automatically starts again with sub-buffer 0.

To collect more data than will fit in memory at one time, an almost identical structure is used. In this case, however, the user program passes the specific number of data points to the DTLIB routine. The number of points to read in a case like this would probably be much larger than the storage space available to store the points. The same sort of handshaking between completion routines/main program/DTLIB code occurs through the buffer event flag. In this case, the completion routine must still empty a certain number of sub-buffers before DTLIB needs them again for re-use or an overrun error will be flagged.

If overrun errors occur using either of the two previous structures, the problem can be partially alleviated by making the sub-buffers larger (allows more time between completion routines).

Each sub-buffer must contain an integral number of data points, although each data point may require more than one IBUF location for storage. For example, the following sub-buffer structure resulted from a call to the RTS routine that specified NCCH=5, meaning that each data-point represented a sweep of five consecutive A/D channels:



The DTLIB argument checking code requires that the following conditions be met in partioning buffers:

1. sub-buffer size will be the integer result of ISIZ/NBUF. ISIZ will then be reduced (if necessary) so that ISIZ=NBUF*sub-buffer size.

2. sub-buffer size must contain an integral number of data-points. Under RTS and ISORTS, this means that (ISIZ/NBUF)/NCCH must result in an integer with no remainder. Under HIST and DRS, a data-point takes either one or two words of storage depending upon the operating mode, so either ISIZ/NBUF = an integer or (ISIZ/NBUF)/2 = an integer.

3. NREAD (if positive and therefore indicating the number of data-points to take) must represent an integral number of sub-buffers. In other words, NREAD divided by the number of data-points in a sub-buffer must result in an integer with no remainder. This means that NREAD/((ISIZ/NBUF)*size of data-point) must be an integer result with no remainder.

If either a buffer-argument error or a sub-buffer argument error occurs, verify that the above requirements have been met.

2.4 Completion Routines

One of DTLIB's most powerful features is the use of completion routines in conjunction with asynchronous sampling. Asynchronous sampling is used by several DTLIB routines, and represents an operation that is initiated at one point in the

user FORTRAN program, but then functions completely independently of the user's program until the operation is complete (see the programming examples at the end of this chapter). The user's FORTRAN program continues to execute as though nothing else was occurring, although the asynchronous sampling is collecting data continuously.

Completion routines represent an important extension to this idea of asynchronous data collection. The user can have DTLIB schedule the execution of completion routines at important points in the data collection process (such as the filling of a sub-buffer with data). When the asynchronous collection process reaches this point, the specified completion routine is automatically executed. The data collection continues even while the completion routine is executing unless instructed to wait by the user.

Before continuing with this discussion, the term "completion routine" must be clarified. A completion routine is a FORTRAN subroutine that may contain any legal FORTRAN sub-program statements. The routine may be long or short, and it may set flags for the main program or process data itself. In short, the user may do virtually anything he wants to do in the completion routine, subject to the constraints below. The routine will not be executed until some DTLIB routine has reached a program-declared important event. The actual mechanism for scheduling completion routines will be described in Chapter 3 when the DTLIB routines are defined and discussed.

There are some restrictions on the use of sub-routines as completion routines:

1. The completion routine must end with a FORTRAN RETURN statement to return control to the DTLIB and RT-11 scheduler.

2. The name of the FORTRAN subroutine must be declared EXTERNAL if it is to be passed to DTLIB for use as a completion routine.

3. If the subroutine is to be used as a completion routine, it may not be called as a regular subroutine from anywhere in the user FORTRAN program. FORTRAN does not support re-entrant subroutines - the user must guarantee that the subroutine has fully completed execution before another entry is made.

4. Because FORTRAN does not support re-entrant routines, the completion routine should never call another subprogram that could also be called simultaneously from the main program (such as user-written functions or subroutines, FORTRAN mathematics subprograms such as SIN, and FORTRAN formatted I/O. If a DTLIB routine is called from within a completion routine, it must either be called in the special MODIFY MODE or that

particular DTLIB routine cannot be also called from the main program.

5. No arguments may be passed directly to or from the completion routine. All arguments or parameters that are used by the completion routine must reside in a FORTRAN COMMON block.

6. Avoid including in a completion routine a request for synchronous input or output from a slow (<9600 baud) terminal. Such a request can cause a significant delay before the completion routine can return control to the DTLIB and RT-11 scheduler.

Although not recommended for general applications, MACRO-11 subroutines may be used for completion routines also. A partial list of requirements for this is as follows:

1. All registers must be saved - the routine cannot return any modifed registers.

2. Any interaction with FORTRAN parameters or variables must be handled by the user (you must be familiar with the run-time environment).

3. The name of the MACRO-11 routine must be declared EXTERNAL in the FORTRAN program.

4. The MACRO-11 code must be separately assembled, then linked to the final FORTRAN program.

Completion routines can be visualized as FORTRAN-coded interrupt handlers. The interrupt is actually a data collection event rather than a hardware interrupt, but the concept is the same: the main program is interrupted, the interrupt handler (completion routine) is executed in response to the service request, and the main program then continues as though nothing had happened at all. It is important for the user to distinguish between the details of the hardware interrupt as opposed to the software completion routine.

Hardware interrupt:

No matter what type of program the processor is currently executing, it always handles the interrupt from a hardware device. The processor will handle all outstanding hardware interrupts before returning to continue execution of any user's program or sub-program.

For example, if the interrupt is from an A/D converter, the processor stops execution of the current program, runs the DTLIB code that stores the A/D data in memory,

then dismisses the interrupt and returns to the original program if no other hardware interrupts are active.

Completion routine interrupt:

When one of the DTLIB code modules that handle hardware interrupts from peripherals determines that a completion routine is necessary (an important collection event has occurred), the DTLIB code queues a request for completion routine execution with the RT-11 monitor. The RT-11 monitor will then handle these queued requests as soon as time permits. There can be several sources of hardware interrupts, and therefore there can be many requests for completion routines, possibly occurring nearly simultaneously. The monitor queues all of these requests and processes them on a time available basis.

The Single Job monitor and the Foreground/Background monitor handle completion routines differently. In the RT-11 SJ monitor, completion routines are totally asynchronous: completion routines can interrupt each other (last-in, first-out arrangement). In the RT-11 FB monitor, completion routines are queued and made to wait until the correct job is running: completion routines cannot interrupt each other (first-in, first-out basis). The order of execution priority is

foreground or background hardware interrupt requests
foreground completion routines
foreground main program
background completion routines
background main program

The user is urged to read the DEC RT-11 literature on completion routines and completion routine scheduling within the monitor for much more detailed information about how the monitor itself processes the completion routine queue.

NOTE

When a completion routine is executing, it executes at the interrupt priority specified in the DTLIB call that is scheduling that completion routine. It is recommended that an interrupt priority of 0 be used for completion routines, as this allows both system and DTLIB interrupts to be serviced while the completion routine is executing. IF HARDWARE INTERRUPTS ARE LOCKED OUT DUE TO THE USE OF A HIGH INTERRUPT PRIORITY, BOTH SYSTEM AND DTLIB DATA TRANSFERS WILL STOP COMPLETELY UNTIL THE COMPLETION ROUTINE COMPLETES EXECUTION.

The primary uses of completion routines within DTLIB are as follows:

1. To process the data now available in the full sub-buffer (scheduled every sub-buffer).

2. To arrange for the last sub-buffer to be transferred to disk or storage (scheduled every sub-buffer).

3. To handle any user defined protocol for synchronizing data collection with the main program (in addition to pre-defined DTLIB structures for synchronization).

4. To collect data continuously or to collect more data than will fit in memory (see the next section on this topic).

5. To determine if sampling should now stop because of some special condition (such as data becoming too large, too small, etc.)

2.5 Sample Programs distributed in release kits

Included with the binary distribution kits are several sample FORTRAN programs that demonstrate the usage of some of the DTLIB routines. Each sample program (the filename extension will be ".FOR") contains lengthy internal comments, describing each part of the program. The user is urged to look these programs over (some consist of several code modules) if additional questions are raised about DTLIB usage. Some of these programs are reproduced in Appendix D of this manual. Check your copy of the distribution kit for these programs.

CHAPTER 3

Descriptions of DTLIB Routines and Functions

3.1 Introduction

This chapter discusses the syntax details of calling DTLIB routines and functions from within a FORTRAN program. There are many subtleties and considerations involved with using these routines in user applications - these considerations are not discussed in this chapter. The user is strongly urged to read the previous two chapters carefully to become aware of the techniques for achieving the desired performance level.

3.2 Conventions

Throughout the remainder of this chapter the following conventions will be followed:

Optional arguments:

Arguments that do not always need to be specified are shown enclosed between square brackets. This indicates that the argument may be left blank. For example,

```
argument present:                ---,IUNIT,---
argument blank (defaulted):      ---,,---
```

In the syntax description, this is shown as ---,[IUNIT],--- . If the commas that separate arguments are also inside the brackets, the commas may be left out also.

Default values:

Where DTLIB substitutes a value for a blank argument, the description of the argument will specify the value that DTLIB will use.

Argument types:

All arguments are of type INTEGER*2 unless otherwise stated. It is important that the proper type of arguments be passed to DTLIB routines or errors in operation may occur.

Passing literals as arguments:

It is permissible to pass literals (read-only parameters) as arguments to DTLIB routines except in argument positions that require read/write variables. If the argument will be written into by DTLIB, the description of the argument will state this. If a literal is passed as one of these read/write arguments, the user program may be corrupted. For example,

```
read/write argument     ---,IUNIT,---
literal argument        ---,7,---
```

3.3 Summary of DTLIB Routines and Functions

The following two tables summarize the names and functions of the DTLIB routines supplied in the distribution kit. The remainder of this chapter will deal exclusively with individual descriptions of each of these routines.

Routine	Description	Hardware supported
IADC	single A/D conversion	DT2762,DT2772 DT2764,DT2774 DT2781,DT2785 DT2782,DT2784
ISOADC	single A/D conversion on an isolated input channel	DT2765,DT2775
RTS	real-time sampling of the analog input channels	DT2762,DT2772 DT2764,DT2774 DT2781,DT2785 DT2782,DT2784
ISORTS	real-time sampling of the isolated analog input channels	DT2765,DT2775
SETR	set clock rate and mode	DT2769
HIST	collect data on Schmitt Trigger #2 events	DT2769
IDAC	modify the value of a single D/A output channel	DT2766,DT2767
IDIR	reading a digital input channel	DT2768,DT2768-I

IDOR	loading a digital output channel	DT2768,DT2768-I
DRS	digital read-in sampling	DT2768,DT2768-I
DPOLL	clock-driven digital input polling	DT2768,DT2768-I
DISP	display data on XY scope or XY plotter	DT2771

Table 3-1 Summary of Peripheral Control Routines

Routine	Description
FLT16	convert unsigned INTEGER*2 to REAL*4
INT16	convert REAL*4 to unsigned INTEGER*2
KBCD2B	convert BCD to binary
KB2BCD	convert binary to BCD
LWAIT	wait for real-time operation complete
CVSWG	convert switch-gain data

Table 3-2 Summary of General Purpose Routines

In addition to the routines described in the above two tables, more routines and functions are continually being developed by Data Translation. The routines documented in this manual are those corresponding to the DTLIB version number on the front cover of this manual.

The following sections in this chapter describe each routine individually.

3.3.1 IADC Single A/D Conversion

The IADC routine initiates a single A/D conversion on a specified analog input unit on a specified channel. This routine is synchronous - that is, the routine will not return to the user's FORTRAN program until the data has been taken.

```
CALL    IADC(ICHAN,IUNIT [,[IGAIN][,IVAR]])
                    or
IX  =   IADC(ICHAN,IUNIT [,[IGAIN][,IVAR]])
```

ICHAN Specifies from which analog input channel the data is to be taken. The channel number may range from 0-63, but attempts to reference channels not installed on a given analog input unit will be reported as errors.

IUNIT Specifies from which analog input unit the data is to be taken. See Chapter 4.

IGAIN If the analog input unit specified contains the programmable gain instrumentation amplifier, IGAIN selects the gain setting. There are two types of software-programmable gain amplifiers: high level and low level. Consult the appropriate hardware User Manual for information on the type of amplifier on your board (if any).

IGAIN	Gain setting high level,PGH	low level,PGL
defaulted	gain = 1	gain = 1
0	gain = 1	gain = 1
1	gain = 1	gain = 1
2	gain = 2	gain = 10
3	gain = 4	gain = 100
4	gain = 8	gain = 500

IVAR If present in the argument list, the A/D data is stored in this INTEGER*2 variable in addition to being returned as the value of the function call. Do not pass a literal in this argument position.

When this routine is called as a function (IX=IADC(....)), the data is returned as the value of the function. If called as a subroutine (CALL IADC(....)), the data can only be stored in the variable described as IVAR above. All data is returned in 16 bit format exactly as received from the A/D converter. Depending upon how the A/D interface is configured, this data will be in 2's complement notation (FORTRAN INTEGER*2 notation), 16 bit offset binary notation, or 16 bit straight binary notation.

3.3.2 ISOADC Single A/D Conversion

The ISOADC routine initiates a single A/D conversion on a specified isolated analog input unit on a specified channel. This routine is synchronous - that is, the routine will not return to the user's FORTRAN program until the data has been taken.

```
CALL    ISOADC(ICHAN,IUNIT [,[IGAIN][,IVAR]])
                    or
IX  =   ISOADC(ICHAN,IUNIT [,[IGAIN][,IVAR]])
```

ICHAN Specifies from which isolated analog input channel the data is to be taken. The channel number may range from 0-60, but attempts to reference channels not installed on a given isolated analog input unit will be reported as errors.

IUNIT Specifies from which isolated analog input unit the data is to be taken. See Chapter 4.

IGAIN If the isolated analog input unit specified contains the programmable gain instrumentation amplifier, IGAIN selects the gain setting.

IGAIN	Gain setting
defaulted	gain = 1
0	gain = 1
1	gain = 1
2	gain = 10
3	gain = 100
4	gain = 500

IVAR If present in the argument list, the A/D data is stored in this INTEGER*2 variable in addition to being returned as the value of the function call. Do not pass a literal in this argument position.

When this routine is called as a function (IX=ISOADC(....)), the data is returned as the value of the function. If called as a subroutine (CALL ISOADC(....)), the data can only be stored in the variable described as IVAR above. All data is returned in 16 bit format exactly as received from the A/D converter. Depending upon how the A/D interface is configured, this data will be in 2's complement notation (FORTRAN INTEGER*2 notation), 16 bit offset binary notation, or 16 bit straight binary notation.

3.3.3 RTS Real-time Sampling of the A/D Converter

The RTS routine initiates and controls real-time sampling of the analog input channels. The A/D converter can be sampled in a variety of modes with the collected data being stored in an input buffer. This routine is asynchronous - it is called once to set up the sampling, then returns to the user's FORTRAN program while the data is being collected via interrupt-driven sampling "underneath" the FORTRAN program. The flag ICMF can be used to monitor the status of the sampling. Once sampling is in progress, RTS can only be called in the special MODIFY mode until sampling has come to a full stop. One exception: FAST SWEEP mode operates synchronously - RTS does not return to the user's program until all data has been collected.

```
CALL    RTS([IBUF],ISIZ,[NBUF],[NREAD],[ISTCHN],
            [NCCH],[IUNIT*],[IGAIN],[MODE*],ICMF,IBEF
            [,[INTCT*][,[CMPRTN*][,[IPRIO*]]]])
```

IBUF — Name of an INTEGER*2 single dimensional array to be used as an input buffer. If this parameter is left blank (defaulted), the call is in the special "MODIFY MODE" and only those parameters followed by an asterisk may be modified. This array will be used to store the A/D data.

ISIZ — Total length (words) of IBUF used for the input buffer. Chapter 2 discusses buffer partioning considerations, and how NBUF and ISIZ are used. This parameters should <u>never</u> be larger than the maximum size of the IBUF array, although it may be less. This argument is used solely to check the legality of NBUF, NREAD, and NCCH.

NBUF — Number of sub-buffers to be maintained. If left blank (defaulted), a value of one will be substituted. NBUF cannot be greater than ISIZ.

Small sub-buffers cause DATA OVERRUN errors when used with fast sampling rates. The bigger the sub-buffer, the smaller the chance of getting DATA OVERRUN errors. The sampling rate (generated externally or by the Real Time Clock) and the size of the sub-buffers are directly related. Faster sampling rates require larger sub-buffers.

NREAD — Number of data points to take. A "data point" consists of all the data read as the result of a sampling trigger. NREAD is the <u>total</u> number of such data points to collect.

The number of INTEGER*2 values read per sampling

trigger equals NCCH (see description of ISTCHN and NCCH below for a more detailed explanation).

ISTCHN Starting channel for the sample. The default value is zero. The channel number may range from 0-63, but attempts to reference channels not installed on a given analog input unit will be reported as errors.

NCCH Number of consecutive channels to be sampled. The default value is one. This value can range from 1 to 64, but its value must not exceed the number of channels installed on the A/D unit. Sampling will wrap-around: after the last channel installed is sampled, the next channel sampled (if more are requested) will be channel 0, etc.

Each external trigger will cause a "data point" to be taken. This data point actually consists of several values: RTS will "sweep" the input channels, starting at ISTCHN and sampling NCCH consecutive channels. If NCCH=1 only a single channel will be sampled.

The total number of values (INTEGER*2 numbers) that will be stored in IBUF is NCCH*NREAD under normal sampling.

IUNIT Specifies from which analog input unit the data is to be taken. See Chapter 4.

IGAIN If the analog input unit specified contains the programmable gain instrumentation amplifier, IGAIN selects the gain setting. There are two types of software-programmable gain amplifiers: high level and low level. Consult the appropriate hardware User Manual for information on the type of amplifier on your board (if any).

IGAIN	Gain setting	
	high level,PGH	low level,PGL
defaulted	gain = 1	gain = 1
0	gain = 1	gain = 1
1	gain = 1	gain = 1
2	gain = 2	gain = 10
3	gain = 4	gain = 100
4	gain = 8	gain = 500

MODE Mode of sampling to be used. The STOP CODES recognized (valid in MODIFY MODE only) are:

Value	Meaning
-1	Halts all sampling and stops all queued completion routine requests.
-2	Halts all sampling and allows the completion

routines that have been queued to finish. It does not queue any more requests.

-3 Completes operation on the current sub-buffer and stops all queued completion routine requests.

-4 Completes operation on the current sub-buffer and allows the completion routines that have been queued to finish. It does not queue any more requests.

Operating modes:
(add codes of all desired functions to generate MODE)

0 Sample upon events sensed by EXTERNAL TRIGGER (EXT TRIGL). Connect TTL trigger source to hardware EXT TRGL.

add 1 to disable BURST MODE operation (applies to FAST SWEEP and DMA modes only). BURST MODE uses the external sampling trigger to start the first conversion, then "sweeps' the rest of the channels (as specified by NCCH) at maximum speed. After the sweep, RTS waits for the next sampling trigger. If BURST MODE is disabled under FAST SWEEP and DMA modes, all conversions will be triggered under external control, regardless of the number of channels to be "swept".

add 2 to use the REAL TIME CLOCK input (RTC INL) as the trigger source rather than the EXTERNAL TRIGGER (EXT TRIGL). Connect TTL trigger source to hardware RTC INL.

add 4 to enable FAST SWEEP mode. In FAST SWEEP mode, the processor is dedicated to collecting data at the maximum possible rate. Whereas the normal mode of collection is interrupt-driven, FAST SWEEP does not use interrupts. See Appendix B for throughput values.

There are some restrictions regarding the use of FAST SWEEP:

NBUF must equal one (single buffer only). IBEF and INTCT will be ignored (IBEF must still be present).

DMA mode cannot be selected simultaneously.

ISTCHN must specify the channel to be sampled. NCCH will be ignored - FAST SWEEP will sample single channels only.

CMPRTN must not be specified - completion routines are illegal in FAST SWEEP mode. Leave CMPRTN blank.

IPRIO specifies the priority of the processor during the sampling interval. If defaulted, PRIORITY 0 will be

used. If interrupts are allowed during the sampling interval, there will a drastic decrease in the maximum sampling rate that can be supported without OVERRUN errors. It is recommended that PRIORITY 6 be specified in FAST SWEEP mode.

add 8 to enable transfers to occur under Direct Memory Access (legal on DMA models only). This mode of data collection will be almost two orders of magnitude faster then the normal collection mode. Because of hardware limitations, MODE 2 must also be specified.

ICMF the completion/error flag. The flag should be set to zero by the user's program before calling RTS. This flag is incremented by one after all points have been read or after RTS is stopped by a stop code.

If an error occurs (either a hardware or software DATA OVERRUN), ICMF is set to a value of -1. A DATA OVERRUN error may result from an insufficient number or size of sub-buffers, or from a hardware sampling rate that is too high.

ICMF is a read/write argument - do not pass a literal in this argument position.

IBEF the buffer event flag. IBEF keeps a count of the number of sub-buffers currently available to store A/D data. The RTS routine initially sets IBEF to the value of NBUF (the number of sub-buffers). After each sub-buffer has been filled, IBEF is decremented by one. If the user is supplying completion routines to process the data in sub-buffers, he should increment IBEF (within the completion routine) to signal that a sub-buffer has become free again.

If the RTS routine decrements IBEF to zero before the last point is read, a DATA OVERRUN error is generated.

IBEF is a read/write argument - do not pass a literal in this argument position.

INTCT the point interrupt count. For every INTCT points gathered, a completion routine (if present) is scheduled. However, this scheduling takes place only at the end of sub-buffers regardless of the size of INTCT. One completion routine execution is scheduled for each INTCT points.

The default value for INTCT corresponds to a request at the end of each sub-buffer. Thus, the default value is equal to ISIZ/(NBUF*NCCH).

CMPRTN the name of the user-supplied completion routine. This

name must be declared EXTERNAL in the FORTRAN program that contains the call to RTS or the FORTRAN compiler will report an error. The completion routine itself may be either a FORTRAN subroutine or a MACRO-11 subroutine.

IPRIO the interrupt priority level to be used by RTS when executing completion routines. The default value is 0 (PRIORITY 0, PR0). Specify this value in a 0-7 range (representing PR0 to PR7 respectively).

3.3.4 ISORTS Real-time Sampling of the Isolated A/D

The ISORTS routine initiates and controls real-time sampling of the isolated analog input channels. The isolated A/D converter can be sampled in a variety of modes with the collected data being stored in an input buffer. This routine is asynchronous - it is called once to set up the sampling, then returns to the user's FORTRAN program while the data is being collected via interrupt-driven sampling "underneath" the FORTRAN program. The flag ICMF can be used to monitor the status of the sampling. Once sampling is in progress, ISORTS can only be called in the special MODIFY mode until sampling has come to a full stop.

```
CALL    ISORTS([IBUF],ISIZ,[NBUF],[NREAD],[ISTCHN],
            [NCCH],[IUNIT*],[IGAIN],[MODE*],ICMF,IBEF
            [,[INTCT*][,[CMPRTN*][,[IPRIO*]]]])
```

IBUF — Name of an INTEGER*2 single dimensional array to be used as an input buffer. If this parameter is left blank (defaulted), the call is in the special "MODIFY MODE" and only those parameters followed by an asterisk may be modified. This array will be used to store the A/D data.

ISIZ — Total length (words) of IBUF used for the input buffer. Chapter 2 discusses buffer partioning considerations, and how NBUF and ISIZ are used. This parameters should <u>never</u> be larger than the maximum size of the IBUF array, although it may be less. This argument is used solely to check the legality of NBUF, NREAD, and NCCH.

NBUF — Number of sub-buffers to be maintained. If left blank (defaulted), a value of one will be substituted. NBUF cannot be greater than ISIZ.

Small sub-buffers cause DATA OVERRUN errors when used with fast sampling rates. The bigger the sub-buffer, the smaller the chance of getting DATA OVERRUN errors. Note, though, that the maximum sampling rates provided by the hardware are in the range of 20 samples/second. Therefore, it should be difficult to generate DATA OVERRUN errors.

NREAD — Number of data points to take. A "data point" consists of all the data read as the result of a sampling trigger. NREAD is the <u>total</u> number of such data points to collect.

The number of INTEGER*2 values read per sampling trigger equals NCCH (see description of ISTCHN and NCCH

below for a more detailed explanation).

ISTCHN Starting channel for the sample. The default value is zero. The channel number may range from 0-60, but attempts to reference channels not installed on a given isolated analog input unit will be reported as errors.

NCCH Number of consecutive channels to be sampled. The default value is one.

Each "data point" taken actually consists of several values: ISORTS will "sweep" the input channels, starting at ISTCHN and sampling NCCH consecutive channels. If NCCH=1 only a single channel will be sampled.

The total number of values (INTEGER*2 numbers) that will be stored in IBUF is NCCH*NREAD. This total must be equal to the value of ISIZ.

IUNIT Specifies from which isolated analog input unit the data is to be taken. See Chapter 4.

IGAIN If the isolated analog input unit specified contains the programmable gain instrumentation amplifier, IGAIN selects the gain setting.

IGAIN	Gain setting
defaulted	gain = 1
0	gain = 1
1	gain = 1
2	gain = 10
3	gain = 100
4	gain = 500

MODE Mode of sampling to be used. The STOP CODES recognized (valid in MODIFY MODE only) are:

Value	Meaning
-1	Halts all sampling and stops all queued completion routine requests.
-2	Halts all sampling and allows the completion routines that have been queued to finish. It does not queue any more requests.
-3	Completes operation on the current sub-buffer and stops all queued completion routine requests.
-4	Completes operation on the current sub-buffer and allows the completion routines that have been queued to finish. It does not queue any

more requests.

Operating modes (add codes of all desired functions)

0 Start sampling immediately, and free-run at converters maximum speed, collecting data according to the other parameters in the call (rate will be 20 conversions/second)

ICMF the completion/error flag. The flag should be set to zero by the user's program before calling ISORTS. This flag is incremented by one after all points have been read or after ISORTS is stopped by a stop code.

If an error occurs (either a hardware or software DATA OVERRUN), ICMF is set to a value of -1. A DATA OVERRUN error may result from an insufficient number or size of sub-buffers, indicating that data is being collected faster than it is being processed.

ICMF is a read/write argument - do not pass a literal in this argument position.

IBEF the buffer event flag. IBEF keeps a count of the number of sub-buffers currently available to store A/D data. The ISORTS routine initially sets IBEF to the value of NBUF (the number of sub-buffers). After each sub-buffer has been filled, IBEF is decremented by one. If the user is supplying completion routines to process the data in sub-buffers, he should increment IBEF (within the completion routine) to signal that a sub-buffer has become free again.

If the ISORTS routine decrements IBEF to zero before the last point is read, a DATA OVERRUN error is generated.

IBEF is a read/write argument - do not pass a literal in this argument position.

INTCT the point interrupt count. For every INTCT points gathered, a completion routine (if present) is scheduled. However, this scheduling takes place only at the end of sub-buffers regardless of the size of INTCT. One completion routine execution is scheduled for each INTCT points.

The default value for INTCT corresponds to a request at the end of each sub-buffer. Thus, the default value is equal to ISIZ/(NBUF*NCCF).

CMPRTN the name of the user-supplied completion routine. This name **must** be declared EXTERNAL in the FORTRAN program that contains the call to ISORTS or the FORTRAN compiler will report an error. The completion routine itself may be either a FORTRAN subroutine or a MACRO-11 subroutine.

IPRIO the interrupt priority level to be used by ISORTS when executing completion routines. The default value is 0 (PRIORITY 0, PR0). Specify this value in a 0-7 range (representing PR0 to PR7 respectively).

3.3.5 SETR Set Clock Rate and Mode

The SETR routine allows the user to select the operating rate and mode of any of the Real Time Clocks in the system. Note that the Line Time Clock in the system is totally separate from the Real-Time Clocks. SETR controls only the Real-Time Clocks.

```
CALL    SETR([IRATE],[IUNIT*],MODE,PRESET,ICMF
             [,[INTCT*][,[CMPRTN*][,[IPRIO*]]]])
```

IRATE the rate for the oscillator which drives the clock counter system. If this parameter is left blank (defaulted), the call is in the special "MODIFY MODE" and only those parameters followed by an asterisk may be modified. If IRATE is present, either an operating rate or a STOP CODE may be specified. The STOP CODES recognized are:

Value	Meaning
-1	Halts the clock and stops all queued completion routine requests.
-2	Halts the clock and allows the completion routines that have been queued to finish. It does not queue any more requests.
-3	Same as -1
-4	Same as -2

Operating rates:

0	Stop Clock (does not function like a stop code). Use this code to enable HIST routine operation if the timing function of the clock is not needed.
1	1 MHz count rate
2	100 KHz count rate
3	10 KHz count rate
4	1 KHz count rate
5	100 Hz count rate
6	Schmitt Trigger #1 input frequency
7	BEVNT line (BUS EVENT) - normally 50/60 Hz

IUNIT Specifies which Real-Time Clock unit is to operated on this call. See Chapter 4.

MODE the mode of clock operation. Select one of the four basic modes from the table below, then add in any additional special functions, also described below.

Basic Modes:

Value	Meaning
0	Single interval mode. Clock will count PRESET intervals, overflow, and stop.
1	Repeated interval mode. Clock will count PRESET intervals, overflow, reload PRESET, and count again. The cycle will continue until the clock is stopped with a STOP CODE. This generates a stable hardware timebase for use by the A/D hardware, and provides a software timebase for routines DRS and DPOLL.
2	External event timing mode. The counter is initialized to zero, then counts at the oscillator rate. A Schmitt Trigger #2 event will transfer the contents of the clock counter into the Buffer/Preset register of the interface (not the PRESET parameter). The clock continues counting, with overflows generating interrupts.
3	Event timing mode from zero base. Similar to mode 2, except that the counter is cleared after transferring the current count to the Buffer/Preset register.

Additional functions:

add 4 starts clock operating upon a Schmitt Trigger #2 event rather than starting immediately.

add 8 disable clock overflow interrupts. Use this mode to generate hardware overflow pulses that will occur too fast for the software interrupt handler to process. In this mode, the 16 bit software clock will not operate, nor will the clock-overflow driven sampling provided by DRS and DPOLL. The DRS routine can, however, be used for digital-interrupt driven sampling.

add 16 low-overhead interrupt handling. This allows faster interrupt rates to be supported, but at the expense of DRS clock-driven sampling and completion routine scheduling. The 16-bit software clock and the DPOLL sampling will still function however.

PRESET the interval count value for clock modes 0 and 1. It must be equal to or less than 65535 (decimal). THIS PARAMETER MUST BE A REAL*4 VALUE.

The clock overflow rate in modes 0 and 1 can be determined by dividing the clock oscillator rate (IRATE) by the PRESET value. Thus, an overflow rate of 500 Hz may be obtained by setting IRATE=3 (1 KHz rate) with PRESET=2.0 (overflow = 1Khz / 2 = 500 Hz).

In clock modes 2 and 3, the value passed as PRESET will not be used but still must be present.

ICMF the completion/error flag. The flag should be set to zero by the user's program before calling SETR. This flag is incremented by one only when the clock overflows in mode 0, or when the clock is stopped by a stop code.

If a CLOCK OVERRUN error occurs (clock rate too fast), ICMF will be set to -1.

ICMF is a read/write argument - do not pass a literal in this argument position.

INTCT the clock-overflow interrupt count. For every INTCT clock interrupts, a completion routine (if present) is scheduled.

The default value for INTCT corresponds to a request after every clock overflow interrupt.

CMPRTN the name of the user-supplied completion routine. This name must be declared EXTERNAL in the FORTRAN program that contains the call to SETR or the FORTRAN compiler will report an error. The completion routine itself may be either a FORTRAN subroutine or a MACRO-11 subroutine.

IPRIO the interrupt priority level to be used by SETR when executing completion routines. The default value is 0 (PRIORITY 0, PR0). Specify this value in a 0-7 range (representing PR0 to PR7 respectively).

Note about the 16 bit software clock:

SETR maintains an internal 16 bit software clock for each real-time clock. This clock is incremented by one after every hardware overflow interrupt. When the clock reaches a value of octal 177777 (decimal 65535, INTEGER*2 -1), it resets to zero and continues to count up. DTLIB routines IDIR and IDOR may be used to set and read this software clock, and the clock may also be read in special modes under the DRS or HIST routines.

3.3.6 HIST Collect Data on Schmitt Trigger #2 Events

The HIST routine allows time-interval data to be obtained from the Real-Time clock, with the Schmitt Trigger #2 as the sampling trigger. Either 16 or 32 bit data may be obtained. The routine also allows sampling memory locations on ST2 events. This routine is asynchronous - it is called once to set up the sampling, then returns to the user's FORTRAN program while the data is being collected via interrupt-driven sampling "underneath" the FORTRAN program. The flag ICMF can be used to monitor the status of the sampling. Once sampling is in progress, HIST can only be called in the special MODIFY mode until sampling has come to a full stop.

```
CALL    HIST([IBUF],ISIZ,[NBUF],[NREAD],[IUNIT*],
             [MODE*],[ISAMPL],ICMF,IBEF
             [,[INTCT*][,[CMPRTN*][,[IPRIO*]]]])
```

IBUF — Name of an INTEGER*2 single dimensional array to be used as an input buffer. If this parameter is left blank (defaulted), the call is in the special "MODIFY MODE" and only those parameters followed by an asterisk may be modified. This array will be used to store the sampled data.

ISIZ — Total length (words) of IBUF used for the input buffer. Chapter 2 discusses buffer partioning considerations, and how NBUF and ISIZ are used. This parameters should <u>never</u> be larger than the maximum size of the IBUF array, although it may be less. This argument is used solely to check the legality of NBUF, NREAD, and MODE.

NBUF — Number of sub-buffers to be maintained. If left blank (defaulted), a value of one will be substituted. NBUF cannot be greater than ISIZ.

Small sub-buffers cause DATA OVERRUN errors when used with fast sampling rates. The bigger the sub-buffer, the smaller the chance of getting DATA OVERRUN errors. The sampling rate (Schmitt Trigger #2 events) and the size of the sub-buffers are directly related. Faster sampling rates require larger sub-buffers.

NREAD — Number of data points to take. A "data point" consists of all the data read as the result of a sampling trigger. NREAD is the <u>total</u> number of such data points to collect.

The number of INTEGER*2 values read per sampling trigger is either one or two depending upon the value of MODE.

IUNIT Specifies from which Real-Time Clock the sampling trigger will originate. See Chapter 4.

MODE Mode of sampling to be used. The STOP CODES recognized (valid in MODIFY MODE only) are:

Value	Meaning
-1	Halts all sampling and stops all queued completion routine requests.
-2	Halts all sampling and allows the completion routines that have been queued to finish. It does not queue any more requests.
-3	Completes operation on the current sub-buffer and stops all queued completion routine requests.
-4	Completes operation on the current sub-buffer and allows the completion routines that have been queued to finish. It does not queue any more requests.

Operating codes:

0	sample Real-Time Clock Buffer/Preset register on ST2 events. If the clock is running in modes 2 or 3, these values represent time-interval data.
1	sample the memory location referenced as ISAMPL on ST2 events. Note that ISAMPL is either a variable that contains the address to be sampled, or is a literal that represents the address to be sampled. If ISAMPL is defaulted, the software clock will be sampled.
add 2	to operate in 2 word mode, where the second word returned is the value of the software clock. In HIST mode 0 operation, this returns 32 bit time data - the high word is the software clock overflow count and the low word is the hardware count. Note that the software clock is not cleared after ST2 events.

ISAMPL represents the memory location to be sampled in HIST mode 1 operation. In HIST mode 0 with the special two word mode enabled (+2), ISAMPL is the value used to preset the software clock before starting ST2 sampling.

ICMF the completion/error flag. The flag should be set to

zero by the user's program before calling HIST. This flag is incremented by one after all points have been read or after HIST is stopped by a stop code.

If an error occurs (either a hardware or software DATA OVERRUN), ICMF is set to a value of -1. A DATA OVERRUN error may result from an insufficient number or size of sub-buffers, or from an ST2 sampling rate that is too high.

ICMF is a read/write argument - do not pass a literal in this argument position.

IBEF the buffer event flag. IBEF keeps a count of the number of sub-buffers currently available to store data. The HIST routine initially sets IBEF to the value of NBUF (the number of sub-buffers). After each sub-buffer has been filled, IBEF is decremented by one. If the user is supplying completion routines to process the data in sub-buffers, he should increment IBEF (within the completion routine) to signal that a sub-buffer has become free again.

If the HIST routine decrements IBEF to zero before the last point is read, a DATA OVERRUN error is generated.

IBEF is a read/write argument - do not pass a literal in this argument position.

INTCT the point interrupt count. For every INTCT points gathered, a completion routine (if present) is scheduled. However, this scheduling takes place only at the end of sub-buffers regardless of the size of INTCT. One completion routine execution is scheduled for each INTCT points.

The default value for INTCT corresponds to a request at the end of each sub-buffer. Thus, the default value is equal to ISIZ/(NBUF*# of words/ data point).

CMPRTN the name of the user-supplied completion routine. This name must be declared EXTERNAL in the FORTRAN program that contains the call to HIST or the FORTRAN compiler will report an error. The completion routine itself may be either a FORTRAN subroutine or a MACRO-11 subroutine.

IPRIO the interrupt priority level to be used by HIST when executing completion routines. The default value is 0 (PRIORITY 0, PR0). Specify this value in a 0-7 range (representing PR0 to PR7 respectively).

3.3.7 IDAC Single D/A Conversion

The IDAC routine reads and modifies the output values of any one of the installed D/A channels. This routine is synchronous - all operations are completed before IDAC returns to the user program.

```
CALL    IDAC(ICHAN,IUNIT [,IDATA])
                  or
IX  =   IDAC(ICHAN,IUNIT [,IDATA])
```

ICHAN Specifies which analog output channel is to be read or written. The channel number may range from 0-3, but attempts to reference channels not installed on a given analog output unit will be reported as errors.

IUNIT Specifies which analog output unit is to be accessed. See Chapter 4.

IDATA If present in the argument list, the contents of IDATA are loaded into the referenced analog output channel and is returned as the value of the function. If IDATA is defaulted (blank), no write operation occurs but rather the current value of the referenced output channel is returned as the value of the function.

IDATA is an INTEGER*2 variable.

When this routine is called as a function (IX=IDAC(....)), the data is returned as the value of the function. If called as a subroutine (CALL IDAC(....)), no data can be returned. All data is in FORTRAN INTEGER*2 two's complement notation regardless of the configuration of the output unit. All necessary data conversions are performed internally.

3.3.8 IDIR Single Reading of a Digital Input Channel

The IDIR routine allows either memory locations or Digital Input Channels to be read by a user program. In addition, IDIR allows the software clock to be read, and also allows the user to perform bit shifts and byte swaps on INTEGER*2 variables. The operation of IDIR is thus either in a Digital Sampling Mode, or a Data Processing Mode. This routine is synchronous - all operations are completed before IDIR returns to the user program.

```
CALL    IDIR([ICHAN],IUNIT [,[MASK][,[MODE][,IVAR]]])
                        or
IX  =   IDIR([ICHAN],IUNIT [,[MASK][,[MODE][,IVAR]]])
```

Digital Sampling Mode

In this mode, either a memory location or a digital input unit is sampled. The sampled data is masked with the argument MASK, and the result is returned as the value of the function call. In addition, the bits from the sampled data that are masked out are checked, and the bit position of the rightmost non-zero bit is returned as the value of IVAR.

To sample memory:

ICHAN must be a positive, non-zero number, and IUNIT must be the even memory address to be used as the sampling location.

To sample a digital input unit:

ICHAN must be zero, and IUNIT must specify the unit number of the digital input unit to be sampled (range of 0-7, see Chapter 4).

Data masking:

The argument MASK is used to select which bits to return as the value of the function. Specifically, IX = <value of register or memory location> .AND. <value of MASK>. If MASK is defaulted, a value of -1 (all bits set) will be substituted.

The remaining bits that were lost in the masking process (that is, <value of register>.AND.(.NOT.MASK)) are examined. If IVAR is present, the bit number (0-15) of the rightmost (or lowest) non-zero bit is returned in IVAR (counting right to left, with -1 indicating that none of the remaining bits were set).

Data Format Conversion:

If the MODE argument is zero or defaulted, the input data is converted from a four digit unsigned Binary Coded Decimal format to a 16 bit straight binary format. If the MODE argument is non-zero, the input data is returned exactly as received and no conversion is performed.

To demonstrate the above, consider this example:

```
IX = IDIR ( 0,0,"1777,1,IVAR )
```

In this example, ICHAN=0, IUNIT=0, MASK= 1777 (base 8), MODE=1, and IVAR is specified. This combination means

* sample the data in digital input unit 0
* use a mask of 0 000 001 111 111 111 (binary), 1777 (octal)
* do not convert from BCD to binary (MODE=1)
* indicate to caller through variable IVAR the highest bit in the digital input unit that was ignored due to masking by 1777

If the digital input unit contains 51724 (octal), then IX will be set to 1724 (octal), which is 51724.AND.1777. IVAR will be set to 12 (decimal), indicating that the rightmost bit in the digital input register that was not also in the mask was bit number 12.

Although this example uses literals in the argument list, variables could just as easily be used. Note however that argument IVAR is a read/write argument - do not pass a literal in this position.

Data Processing Mode

In this mode, the value passed in the MASK argument position is processed according to the value of the MODE argument. To enable this mode of IDIR operation,

* ICHAN must be defaulted or equal to zero.
* IUNIT must be negative

(note: the actual value of IUNIT has meaning in certain circumstances - see below).

MODE is zero:

The value passed as MASK is byte-swapped (that is, the high and low bytes of the 16 bit value are swapped). The result is returned as the value of the function and also as thealue of IVAR, if given.

MODE is positive:

The value of MASK will be bit-shifted right MODE times. Zeroes will be shifted in to fill the word, and bits shifted out are lost. If IVAR is given, .TRUE. (integer value -1) is returned in IVAR if the last bit shifted out was a one, .FALSE. (integer value 0) will be returned if the last bit was a zero. The actual shifted value is returned as the value of the function.

MODE is negative:

The value of MASK will be bit-shifted left MODE times. Zeroes will be shifted in to fill the word, and bits shifted out are lost. If IVAR is given, .TRUE. (integer value -1) is returned in IVAR if the last bit shifted out was a one, .FALSE. (integer value 0) will be returned if the last bit was a zero. The actual shifted value is returned as the value of the function.

MODE is defaulted:

The software clock of clock unit #<(-IUNIT)-1> is returned as the value of the function, and is also returned in IVAR if given. The clock value is ANDed with MASK before being returned. The software clock itself is not modified.

To demonstrate the above, consider the following example:

```
IX = IDIR (,-1,IDATA,4,IVAR)
```

In this example, ICHAN is defaulted, IUNIT=-1, MASK=IDATA, MODE=4, and IVAR is specified. This combination means

* data processing mode
* use value of IDATA
* shift right 4 bits (MODE= +4)
* return status of last bit shifted (IVAR specified)

If IDATA=1763 (octal), 1011 (decimal), then IX will be set to 77 (octal), 63 (decimal). IVAR will be set to .FALSE. as the last bit shifted out was a zero.

Although this example uses literals in the argument list, variables could just as easily be used. Note however that argument IVAR is a read/write argument - do not pass a literal in this position.

3.3.9 IDOR Single Loading of a Digital Output Channel

The IDOR routine allows either memory locations or Digital Output Channels to be loaded by a user program. In addition, IDOR allows the software clock to be read and modified. The operation of IDIR is thus either in a Digital Loading Mode, or a Clock Data Processing Mode. This routine is synchronous - all operations are completed before IDOR returns to the user program.

```
CALL    IDOR([ICHAN],IUNIT [,[MASK][,[IWORD][,IVAR]]])
                          or
IX  =   IDOR([ICHAN],IUNIT [,[MASK][,[IWORD][,IVAR]]])
```

Digital Loading Mode

In this mode, either a memory location or a digital output unit is loaded. The output data is a function of MASK and IWORD.

To write memory:

ICHAN must be a positive, non-zero number, and IUNIT must be the even memory address to be written into.

To load a digital output unit:

ICHAN must be zero, and IUNIT must specify the unit number of the digital output unit to be loaded (range of 0-7, see Chapter 4).

Data masking:

The argument MASK is used to select which bits in the output destination (either memory or a digital output unit) are to be modified. If an output register bit is selected by the presence of a one in MASK at the same position, then the new state of the output bit is determined by the same bit in IWORD. Specifically, for a given bit position (numbers represent the value of the bit, not the position),

MASK bit	IWORD bit	Destination bit
0	don't care	same state as before, not changed
1	0	set to 0
1	1	set to 1

A logical expression that describes this process is presented below:

(IWORD.AND.MASK).OR.((.NOT.MASK).AND.(old value in register))

The default value for MASK if MASK is left blank is -1 (all bits set). The value substituted for IWORD if IWORD is defaulted is zero.

The value returned as the value of the function is the final composite data written to the output destination. If IVAR is present, this data is also returned in IVAR.

Clock Data Processing Mode

In this mode, the software clock of unit number <(-IUNIT)-1> is read and possibly modified (see below). To enable this mode of IDOR operation,

* ICHAN must be defaulted or equal to zero.
* IUNIT must be negative

The software clock of clock unit #<(-IUNIT)-1> is returned as the value of the function, and is also returned in IVAR if given. The clock value is ANDed with MASK before being returned.

After being read, the clock is loaded with the 16-bit value of IWORD. If IWORD is defaulted, the clock will not be loaded. Thus, IDOR can read and write the clock, returning the old value as the value of the function and in IVAR (if present), while loading a new value into the clock.

3.3.10 DRS Real-time Sampling of the Digital Inputs

The DRS routine initiates and controls real-time sampling of the digital input channels. The digital input channels can be sampled in a variety of modes with the collected data being stored in an input buffer. This routine is asynchronous - it is called once to set up the sampling, then returns to the user's FORTRAN program while the data is being collected via interrupt-driven sampling "underneath" the FORTRAN program. The flag ICMF can be used to monitor the status of the sampling. Once sampling is in progress, DRS can only be called in the special MODIFY mode until sampling has come to a full stop.

```
CALL    DRS([IBUF],ISIZ,[NBUF],[NREAD],ICHAN,IUNIT,
            MASK*,[MODE*],ICMF,IBEF
            [,[INTCT*][,[CMPRTN*][,[IPRIO*]]]])
```

IBUF Name of an INTEGER*2 single dimensional array to be used as an input buffer. If this parameter is left blank (defaulted), the call is in the special "MODIFY MODE" and only those parameters followed by an asterisk may be modified. This array will be used to store the digital data.

ISIZ Total length (words) of IBUF used for the input buffer. Chapter 2 discusses buffer partioning considerations, and how NBUF and ISIZ are used. This parameters should <u>never</u> be larger than the maximum size of the IBUF array, although it may be less. This argument is used solely to check the legality of NBUF, NREAD, and MODE.

NBUF Number of sub-buffers to be maintained. If left blank (defaulted), a value of one will be substituted. NBUF cannot be greater than ISIZ.

Small sub-buffers cause DATA OVERRUN errors when used with fast sampling rates. The bigger the sub-buffer, the smaller the chance of getting DATA OVERRUN errors. The sampling rate (generated externally or by the Real Time Clock) and the size of the sub-buffers are directly related. Faster sampling rates require larger sub-buffers.

NREAD Number of data points to take. A "data point" consists of all the data read as the result of a sampling trigger. NREAD is the <u>total</u> number of such data points to collect.

The number of INTEGER*2 values read per sampling trigger is either one or two (see MODE description

below).

ICHAN When ICHAN is zero, IUNIT is interpreted as a unit number to specify which DT2768 unit is to be sampled (see Chapter 4). When ICHAN is a positive number, IUNIT is interpreted as a memory address to be sampled.

IUNIT The meaning of this parameter depends upon the value of ICHAN above. Note that when IUNIT is used to specify a memory address, IUNIT must be an even number.

MASK the 16-bit value that is logically ANDed with the sample data just before the data is stored in IBUF.

MODE Mode of sampling to be used. The STOP CODES recognized (valid in MODIFY MODE only) are:

Value	Meaning
-1	Halts all sampling and stops all queued completion routine requests.
-2	Halts all sampling and allows the completion routines that have been queued to finish. It does not queue any more requests.
-3	Completes operation on the current sub-buffer and stops all queued completion routine requests.
-4	Completes operation on the current sub-buffer and allows the completion routines that have been queued to finish. It does not queue any more requests.

Operating modes:
(add codes of all desired functions to generate MODE)

0 Input data is in Binary-Coded-Decimal format (BCD). Convert to straight binary before storing data.

1 Input data is already in straight binary format. Store data exactly as received.

2 Similar to mode 0 (BCD conversion), except that a second word is also stored in IBUF immediately after the data (this second word is described below).

3 Similar to mode 1, except that a second word is also stored in IBUF immediately after the data (this second word is described below).

The second word stored in modes 2 and 3 is an integer with a value from 0 to 15. This value represents the bit position of

the right-most non-zero bit in the data word (note that the input data is ANDed with MASK before this calculation is done). If a -1 is returned as this value, it indicates that there were no set bits in the masked data.

Unless otherwise specified, triggering will occur under DT2768 REQUEST A control.

add 4 to use REQUEST B as the trigger source rather than REQUEST A.

add 16 to select clock overflow triggering (over-rides the REQUEST B or REQUEST A software options)

In addition, if basic modes 0 or 1 have been selected, the user can instruct DRS to return a "time-stamp" with each collected point. This "time-stamp" is simply the value of one of the software clocks that count clock overflows. The clock value is stored immediately after the collected data.

add 8 to have the software clock returned as the second word

If an operation is selected that uses one of the clocks, DRS must be told which clock to use. The following table indicates what to add to the MODE value to select one of the clocks for either overflow-triggering or for the software clock value.

Add	Clock Unit
nothing	Unit 0
32	Unit 1
64	Unit 2
96	Unit 3

Additional options:

add 128 to have the DT2768 CSR0 output bit set during the sampling interval (useful as a sampling gate to user hardware)

add 256 to have the DT2768 CSR1 output bit set during the sampling interval (useful as a sampling gate to user hardware)

ICMF the completion/error flag. The flag should be set to zero by the user's program before calling DRS. This flag is incremented by one after all points have been read or after DRS is stopped by a stop code.

If an error occurs (either a hardware or software DATA OVERRUN), ICMF is set to a value of -1. A DATA OVERRUN error may result from an insufficient number or size of sub-buffers, or from a hardware sampling rate that is too high.

ICMF is a read/write argument - do not pass a literal in this argument position.

IBEF the buffer event flag. IBEF keeps a count of the number of sub-buffers currently available to store digital data. The DRS routine initially sets IBEF to the value of NBUF (the number of sub-buffers). After each sub-buffer has been filled, IBEF is decremented by one. If the user is supplying completion routines to process the data in sub-buffers, he should increment IBEF (within the completion routine) to signal that a sub-buffer has become free again.

If the DRS routine decrements IBEF to zero before the last point is read, a DATA OVERRUN error is generated.

IBEF is a read/write argument - do not pass a literal in this argument position.

INTCT the point interrupt count. For every INTCT points gathered, a completion routine (if present) is scheduled. However, this scheduling takes place only at the end of sub-buffers regardless of the size of INTCT. One completion routine execution is scheduled for each INTCT points.

The default value for INTCT corresponds to a request at the end of each sub-buffer. Thus, the default value is equal to ISIZ/(NBUF*# of words/data point).

CMPRTN the name of the user-supplied completion routine. This name must be declared EXTERNAL in the FORTRAN program that contains the call to DRS or the FORTRAN compiler will report an error. The completion routine itself may be either a FORTRAN subroutine or a MACRO-11 subroutine.

IPRIO the interrupt priority level to be used by DRS when executing completion routines. The default value is 0 (PRIORITY 0, PR0). Specify this value in a 0-7 range (representing PR0 to PR7 respectively).

3.3.11 DPOLL Digital Input Polling

The DPOLL routine allows either memory locations or digital input units to be monitored for positive bit transitions (change from a 0 to a 1) every clock overflow interrupt. These transitions are recorded for examination later by user software, allowing transient events to be captured. This routine can also be used to 'debounce' digital inputs in this same manner. Finally, DPOLL allows time-latency measurements to be taken - to measure the number of clock-overflow interrupts that occur before a specific digital event occurs.

The DPOLL routine is used in conjunction with the SETR routine. Only one memory location or digital input unit can be polled at a time by a specific clock, but all clock units (up to the maximum of four) can be performing polling at once.

The order of calling would be DPOLL to set-up polling, then SETR to initiate sampling. SETR should be called in repeated-interval mode, with either regular or low-overhead interrupts. Once polling has started, additional calls can be made to DPOLL to determine the status of polling.

```
CALL    DPOLL ( ICLOCK, MODE [ , MASK ] )
                    or
IX = IDPOLL ( ICLOCK, MODE [ , MASK ] )
```

ICLOCK Specifies which Real-Time Clock is to be used in this DPOLL call. This value can range from 0-3, but must represent an installed clock.

MODE Specifies what operation is to be done, and how MASK is to be processed (if present) according to the following:

Polling Set-up

0 clear previous polling operation (if any), interpret MASK as a digital I/O unit number (range 0-7) to be polled.

1 clear previous polling operation (if any), interpret MASK as a memory address (must be an even number) to be polled.

Polling Start

2 start input polling. MASK will be used as a 16 bit mask to determine which bits to examine during the polling operation. The response register is cleared before polling starts.

3 start latency measurement. MASK will be used as a 16 bit mask to determine which bits to examine during the latency measurement. The response register is cleared before polling starts.

Sample Polling Results

4 read results of polling from response register. Use MASK to select which bits to test in the response register, but do not reset the bits that have been tested after testing.

5 similar to (4), but reset the bits that have been tested so that new events can be detected.

6 sample response register to check for a response

7 sample latency register to determine how long the latency time was in real-time clock "ticks".

Stop Polling

8 stop polling operation. MASK ignored.

MASK optional INTEGER*2 argument that is used in certain modes by DPOLL. See the above descriptions for information.

Note that the response register will only detect 0 to 1 transitions of the bits that are being monitored. When a 0-1 transition is detected, the equivalent bit is set in the response register. The response bit will remain set until explicitly cleared, either by starting a new polling operation or by sampling the response register in the "re-arm" mode (mode 5 above). This structure allows brief events to be captured, in addition to insuring that very long events are only reported once. Remember that all polling is done on the basis of the clock interrupts generated by one of the real-time clocks through the SETR routine.

3.3.12 FLT16 Convert to REAL*4

The FLT16 will convert an unsigned INTEGER*2 "time" value such as those produced by IDIR,IDOR,DRS, and HIST from the software clock, and convert it to a REAL*4 value in the 0-65535 (decimal) range.

```
X = FLT16(I)
```

X is a REAL*4 variable and I is an INTEGER*2 variable. An error will be generated if I is defaulted.

3.3.13 INT16 Convert to INTEGER*2

The INT16 will convert a REAL*4 value in the 0.0 to 65535.0 (decimal) range into an unsigned INTEGER*2 value.

```
I = INT16(X)
```

X is a REAL*4 variable and I is an INTEGER*2 variable. An error will be generated if X is defaulted. If X is less than zero, INT16 will return 0. If X is greater than 65535.0, INT16 will return 65535.0.

3.3.14 KBCD2B Convert BCD to Binary

The KBCD2B routine will convert a 4 digit Binary Coded Decimal number stored in an INTEGER*2 variable into straight decimal in the 0-9999 range.

```
I = KBCD2B(J)
```

Both I and J are INTEGER*2 variables. An error will be generated if J is defaulted. Invalid results will be returned if J is not in BCD format.

3.3.15 KB2BCD Convert Binary to BCD

The KB2BCD routine will convert a straight binary number in the 0-9999 range into a four digit Binary Coded Decimal number.

```
J = KB2BCD(I)
```

Both I and J are INTEGER*2 variables. An error will be returned if I is defaulted, if the input data is negative, and if the input data is larger than 9999.

3.3.16 LWAIT Wait for Real-time Operation Complete

The LWAIT routine waits for the value of the first argument to become different from the value of the second argument. The routine will not return to the user program until the arguments are different. When called as a function, the value of the first argument is returned as the value of the function.

```
CALL    LWAIT (I,J)
          or
IX  =   LWAIT (I,J)
```

Both I and J are INTEGER*2 variables. An error will be returned if I or J is defaulted.

3.3.17 CVSWG Convert Switch-Gain Data

The CVSWG routine is provided solely for compatability with earlier versions of DTLIB. The routine converts the INTEGER*2 A/D data into a REAL*4 value. The IGAIN value is ignored. For new programs, the user is urged to use the FORTRAN function FLOAT(IDATA) instead.

```
RDATA = CVSWG (IDATA [,IGAIN])
```

Both IDATA and IGAIN are INTEGER*2 variables. RDATA is a REAL*4 variable. An error is returned if IDATA is defaulted.

3.3.18 DISP Display Data

The DISP routine displays data on an XY oscilloscope or an XY plotter. Data can be continuously refreshed (for an oscilloscope), or be output just once (for a hardcopy plotter).

```
CALL    DISP(MODE [,IBUF1,[IBUF2],IBUF0,
             NPTS,NSTART,INCR])
```

MODE — Mode of display to be used. Select one of the basic codes, then add in any extra options desired. Note: Options can not be added to the -1 code, the stop code.

Value	Meaning
-1	Halts any display of data currently in progress (remainder of arguments ignored).
0	Use the data in IBUF1 as x-channel data, the data in IBUF2 as Y-channel data, generate composite internal-format data in IBUF0, then output a series of X-Y data values to the X and Y D/A converters.
1	Use the data in IBUF1 as X-channel data, ignore IBUF2, generate internal-format data in IBUF0, then output a series of X data values to the X channel D/A converter.
2	Use the data in IBUF1 as Y-channel data, ignore IBUF2, generate internal-format data in IBUF0, then output a series of Y data values to the Y channel D/A converter.
add 8	To display data once rather than continuously (FLASH).
add 16	To disable the internal software triggering of the plotting process. Use this mode if the interface is configured for external control of point-plotting speed (for example, if a hard copy plotter is being used).

IBUF1 — INTEGER*2 array of 12 bit two's complement data representing either X or Y channel data as indicated by MODE. There are NSTART+(NPTS*INCR) words in this array.

IBUF2 INTEGER*2 array of 12 bit two's complement data representing Y channel data (MODE 0 only). Ignored (leave defaulted) in any mode other than mode 0. There are NSTART+(NPTS*INCR) words in this array.

IBUF0 INTEGER*2 array used by DISP to buffer data for the point plotter. This array must be of an appropriate size:

MODE 0:	NPTS*2 words
MODE 1:	NPTS words
MODE 2:	NPTS words

NPTS Number of points to plot. Note that this does not necessarily indicate the size of the input arrays: NSTART+(NPTS*INCR) indicates how many real points are contained in the input buffers - NPTS merely indicates how many of these points will actually be displayed.

NSTART Number of first point to plot in input buffers. This parameter is identical to the FORTRAN array reference - a value of 1 indicates the first point.

INCR Increment between points. For example, INCR=3 means plot every third point. INCR=1 means plot all points.

CHAPTER 4

System Integration

4.1 Introduction

This chapter discusses in detail the procedures required to cause DTLIB to operate properly given a particular peripheral configuration. However, there are some limitations on which DTI interface models and which etch revisions will function completely correctly under DTLIB control. Before attempting to configure and operate DTLIB with your set of interfaces, check the revision level of the interface products to insure that they meet the minimum levels specified in Table 4.1. The interface may have a higher revision level - these are the minimum levels.

Model No.	EP No.	Revision Level	ECO #
DT2762	EP-050	E	146
DT2762/57xx	EP-050	F	177
DT2764	EP-050	E	146
DT2765	EP-050	E	146
DT2766	EP-043	D	153
DT2767	EP-043	D	153
DT2768	EP-058	C	167
DT2768-I	EP-058	C	167
DT2769	EP-057	J	273
DT2771	EP-075	E	270
DT2772	EP-089	B	196
DT2774	EP-089	B	196
DT2775	EP-089	B	196
DT2781	EP-074	C	203
DT2782	EP-073	E	223
DT2784	EP-073	E	223
DT2785	EP-074	B	184

Table 4.1 Minimum Hardware Revision Levels

The Data Translation EP number and the etch revision level will be found on the solder side of the interface circuit card

(the side that does not have components protruding). Should the interface system not meet the revision levels above, consult Data Translation. In many cases, interfaces can either be operated at lower revision levels at some particular sacrifice in DTLIB performance, or can be field-upgraded to current revision levels. The ECO (Engineering Change Order) number listed above next to the etch revision level is the ECO that upgrades earlier revisions to the new etch level. Note: Contact DTI for the correct ECO to field- or factory-upgrade earlier revisions. The ECO numbers listed above are those that upgraded the printed circuit board etch itself.

4.2 Inter-board Connections

Most of the A/D interfaces supported by DTLIB allow a software selectable trigger source. Many users hardwire the two A/D trigger inputs to the outputs of the Real Time Clock. Ordinarily the clock overflow output is connected to the A/D RTC INL input, while the clock ST2 or ST1 output is connected to the A/D EXT TRGL input. The Schmitt Trigger signal conditioning circuitry on the Real Time Clock can be used to generate a TTL output pulse to trigger the A/D when a certain threshold and slope polarity is seen on an analog signal. Consult the DT2769 User's Manual for more information on the operation of the Schmitt Triggers. DTLIB does not require such interconnections: the only requirement is that a trigger source of some kind be connected to the appropriate A/D interface when DTLIB enables the particular trigger input.

4.3 Physical Placement of Interfaces

Because the LSI-11 determines interrupt priority of simultaneous interrupt requests by the location of the interrupting interfaces on the bus, the user can tailor the interrupt response of his system by judicious arrangement of interfaces.

By ranking all of a user's interfaces in order of importance, an interrupt priority can be established. For example, if the most important task in the machine is the collection of data from a particular A/D interface, placing this interface immediately adjacent to the processor card insures that this interface will take priority over all other simulataneous interrupts. This small change in system configuration can allow higher data rates to be achieved.

As a further example, consider a case where the serial console interface is placed in a higher priority bus slot than the A/D interface. Now, if both interfaces request service at

the same time, the processor will handle the serial interface first, temporarily ignoring the A/D request (lower priority). Handling the serial interface request first may result in an A/D data overrun error if another A/D trigger occurs before the previous data was read by the processor.

In summary, higher data acquisition performance will be achieved if the data acquisition interfaces are placed in higher priority bus slots than slower, less important interfaces (printers, serial interfaces, etc.). The user is urged to read the DEC microcomputer literature on interrupts and the LSI-11 bus for a more thorough understanding of system layout.

NOTE

If your system contains a DLV11J, verify that its ECO level is Rev E or above. Rev D and earlier boards may randomly interact with I/O and interrupt transactions regardless of whether the DLV11J is being addressed.

4.4 Installing DTLIB

The DTLIB binary distribution kit consists of a DEC RX01 (single density) compatible diskette (LAB-DATAX software distribution diskettes contain the same files, but are distributed on RX02 double density diskettes). A directory of the files contained on this distribution diskette will be found in Appendix A. After the user verifies that the interface hardware to be supported by DTLIB is all of the correct hardware revision level, the DTLIB package must be installed. The following sections should be read and the instructions obeyed as the user configures a working DTLIB software library customized for his system.

4.4.1 Copying the distribution diskette

Set the DTLIB distribution diskette aside for a moment, and prepare a new blank diskette by typing

INITIALIZE/BADBLOCKS <device where new diskette is located>

This initializes the directory of the diskette and also scans the entire diskette surface for magnetic defects. Now, using the DTLIB distribution disk and the freshly prepared blank diskette, copy the entire contents of the DTLIB diskette

onto the blank diskette. Save the distribution diskette in a safe place as a file copy. Use the newly copied diskette to generate the operating version of DTLIB.

4.4.2 Initializing the hardware configuration file

On the diskette containing the copied DTLIB files, use EDIT, TECO, or KED to create a text file named "CONFIG.MAC" on the diskette. This text file will eventually be used as input to the RT-11 MACRO assembler - if the user desires to insert comments in this file, each comment line must be preceded by a semicolon. Appendix C contains a sample CONFIG.MAC file for use as an example.

This text file must start with

```
.MCALL  START
START
```

These two lines should be at the very beginning of the text file. After these lines, each class of hardware interfaces supported by DTLIB will be described by lines of text. Do not mix types of entries; that is, do not mix classes of interfaces. All analog input interfaces must be described as a group, all clocks as a group, etc.

NOTE

The configuration process automatically assigns software unit numbers to interfaces in a specific group in the order in which multiple interfaces are declared. These unit numbers (described as IUNIT in Chapter 3), start from a zero-base. Therefore, the first interface in a group is unit # 0, the second # 1, and so forth.

4.4.3 Describing analog input interfaces

The analog input interfaces (models DT2762, DT2764, DT2782, DT2784 and the expanders DT2772, DT2774) along with the input portions of the analog I/O interfaces (models DT2781, DT2785) are described by an entry in the "CONFIG.MAC" file in the following format:

```
AINPUT BASE=<bb>,VECTOR=<vv>,NUMCH=<nn>
```

where <bb> is the octal base address of the interface, <vv> is

the octal vector address of the interface, and <nn> is the octal number of installed A/D input channels (including any expander interfaces). All three of these parameters must be correct for the particular interface being described or improper operation will result.

If the particular interface being described has the programmable gain amplifier option, add

,PG=PRESENT

to the end of the AINPUT line. If the interface being described is a model DT2782 or DT2784 with on-board DMA interface, add

,DMA=PRESENT

to the end of the AINPUT line.

NOTE

Models that have both DMA and PG options lose the ability (under DTLIB control) to scan multiple channels and to operate in burst mode (applies only to RTS calls that request the use of DMA).

The first AINPUT entry is assigned software unit number 0, the second entry unit number 1, the third number 2, and the fourth number 3. These unit numbers will be needed by user FORTRAN programs to tell DTLIB which of the analog input interfaces to use when performing A/D conversions. The user may want to put small adhesive labels on the interfaces themselves and/or on their connecting cables to allow easy mapping from the software unit numbers to the actual interfaces. There is a maximum of four entries describing analog input interfaces - any additional entries will result in errors when the user attempts to build his DTLIB library.

If there are no analog input interfaces in the target system, do not enter any AINPUT declarations.

4.4.4 Describing isolated analog input interfaces

The isolated analog input interfaces (model DT2765 and expander DT2775) are described by an entry in the "CONFIG.MAC" file in the following format:

ISOLATED BASE=<bb>,VECTOR=<vv>,NUMCH=<nn>

where <bb> is the octal base address of the interface, <vv> is the octal vector address of the interface, and <nn> is the octal number of installed A/D input channels (including any expander interfaces). All three of these parameters must be correct for the particular interface being described or improper operation will result.

If the particular interface being described has the programmable gain amplifier option, add

,PG=PRESENT

to the end of the ISOLATED line.

The first ISOLATED entry is assigned software unit number 0, the second entry unit number 1, the third number 2, and the fourth number 3. These unit numbers will be needed by user FORTRAN programs to tell DTLIB which of the isolated analog input interfaces to use when performing A/D conversions. The user may want to put small adhesive labels on the interfaces themselves and/or on their connecting cables to allow easy mapping from the software unit numbers to the actual interfaces. There is a maximum of four entries describing isolated analog input interfaces - any additional entries will result in errors when the user attempts to build his DTLIB library.

If there are no analog input interfaces in the target system, do not enter any ISOLATED declarations.

4.4.5 Describing analog output interfaces

There are three distinct forms of the AOUTPUT declaration, depending upon whether a model DT2781/DT2785 (output portion of analog I/O), a model DT2766 (multiple outputs), or a model DT2767 (multiple outputs) analog output interface is being defined.

For a DT2781/DT2785:

AOUTPUT BASE=<bb>,NUMCH=2,TYPE=<tt>

where <bb> is the octal base address of the analog I/O interface plus 4, <tt> is either the word UNIPOLAR or BIPOLAR as appropriate for the interface.

For a DT2766:

AOUTPUT BASE=<bb>,NUMCH=<nn>,BITNUM=TWELVE,TYPE=<tt>

where <bb> is the octal base address of the analog output interface, <nn> is the number of installed output channels

(usually 4), and <tt> is either the word UNIPOLAR or BIPOLAR as appropriate for the interface.

For a DT2767:

AOUTPUT BASE=<bb>,NUMCH=<nn>,BITNUM=EIGHT,TYPE=<tt>

where <bb> is the octal base address of the analog output interface, <nn> is the number of installed output channels (usually 4), and <tt> is either the word UNIPOLAR or BIPOLAR as appropriate for the interface.

The first AOUTPUT entry is assigned software unit number 0, the second entry unit number 1, and so forth up to the eighth entry which is assigned the number 7. These unit numbers will be needed by user FORTRAN programs to tell DTLIB which of the analog output interfaces to use when modifying an output channel. The user may want to put small adhesive labels on the interfaces themselves and/or on their connecting cables to allow easy mapping from the software unit numbers to the actual interfaces. There is a maximum of eight entries describing analog output interfaces - any additional entries will result in errors when the user attempts to build his DTLIB library.

If there are no analog output interfaces in the target system, do not enter any AOUTPUT declarations. Please note that all channels on an interface system must be configured UNIPOLAR or BIPOLAR together - DTLIB does not support mixed channel types on a single interface, only on multiple interfaces.

4.4.6 Describing point plotter interfaces

The point plotter interface (model DT2771) is described by an entry with the following form:

PLOTTER BASE=<bb>,VECTOR=<vv>

where <bb> is the octal base address of the interface, and <vv> is the octal vector address of the interface.

If there are no point plotter interfaces in the target system, do not enter any PLOTTER declarations. Only one PLOTTER entry will be allowed.

4.4.7 Describing real-time clock interfaces

The Real Time Clock interface (model DT2769) is described by an entry with the following form:

```
CLOCK BASE=<bb>,VECTOR=<vv>
```

where <bb> is the octal base address of the interface, and <vv> is the octal vector address of the interface.

The first CLOCK entry is assigned software unit number 0, the second entry unit number 1, the third unit number 2, and the fourth entry unit number 3. These unit numbers will be needed by user FORTRAN programs to tell DTLIB which of the Real Time Clocks to use in a given operation. The user may want to put small adhesive labels on the interfaces themselves and/or on their connecting cables to allow easy mapping from the software unit numbers to the actual interfaces. There is a maximum of four entries describing real time clock interfaces - any additional entries will result in errors when the user attempts to build his DTLIB library.

If there are no Real Time Clock interfaces in the target system, do not enter any CLOCK declarations.

4.4.8 Describing digital I/O interfaces

The digital I/O interfaces (models DT2768 and DT2768-I) are described by entries that take the following form:

```
DIGITAL BASE=<bb>,VECTOR=<vv>
```

where <bb> is the octal base address of the digital I/O interface, and <vv> is the octal vector address of the interface.

If memory sampling is to be used under DRS control, a special pseudo-digital unit must be declared before actual hardware is described. In addition, this pseudo-unit may be declared even if no actual digital hardware is installed to allow DRS to sample memory under clock control. The form of this special declaration is:

```
DIGITAL MEMORY=ACTIVE
```

This declaration must precede any actual digital hardware declarations, but the declaration will not affect the software unit numbering of later declarations.

The first DIGITAL entry is assigned software unit number 0, the second entry unit number 1, and so forth up to the

eighth entry which is assigned the number 7. These unit numbers will be needed by user FORTRAN programs to tell DTLIB which of the digital I/O interfaces to use when modifying an output channel or reading an input channel. The user may want to put small adhesive labels on the interfaces themselves and/or on their connecting cables to allow easy mapping from the software unit numbers to the actual interfaces. There is a maximum of eight entries describing digital I/O interfaces - any additional entries will result in errors when the user attempts to build his DTLIB library.

If there are no digital I/O interfaces in the target system, do not enter any DIGITAL declarations.

4.4.9 Completing the configuration file

When all the hardware has been declared via entries, the following two lines should be inserted at the very end of the file "CONFIG.MAC".

```
DONE
.END
```

Once these lines are included, the file should be closed and the editor exited.

4.4.10 Generating the DTLIB product

With the file "CONFIG.MAC" filled with the correct hardware description entries, the user can now build his customized copy of DTLIB.

To do this, the following logical device assignments must be made:

```
ASSIGN <generation diskette device name> INP
ASSIGN <DTLIB object copy destination device name> OUP
```

Before continuing with the build procedure, the user must make sure that the RT-11 utility programs MACRO and LIBR are present on the system disk. With this accomplished, type

```
@INP:BUILD
```

to execute the command file that will build the DTLIB copy. If any errors are reported, the user should check his file CONFIG.MAC for correctness and try the build procedure again.

The final result of this installation procedure is a file

named DTLIB.OBJ. This file contains the entire DTLIB library, configured for the user's specific peripherals. Once installed, this DTLIB.OBJ file is the only file needed to utilize DTLIB, and is only needed when linking FORTRAN programs (it need not be present when programs are actually run). Save both the original distribution diskette and the backup copy made earlier in a safe place as a precaution against accidental physical or magnetic damage. DTLIB modifications and corrections distributed by Data Translation will require the files found on these two copies.

APPENDIX A

Binary Distribution Kit Contents

The following directory indicates the files distributed with DTLIB V02-01.

```
 03-Sep-80
BUILD .COM      3P 03-Sep-80      CONLIB.ML      18P 03-Sep-80
EXMPL1.FOR     11P 03-Sep-80      EXMPL2.FOR      7P 03-Sep-80
EXMPL3.FOR      6P 03-Sep-80      EXMPL4.FOR      9P 03-Sep-80
EXMPL5.FOR     12P 03-Sep-80      SAVEM .FOR      6P 03-Sep-80
ARG   .OBJ      1P 03-Sep-80      ABLK  .OBJ      2P 03-Sep-80
CMPRTN.OBJ      2P 03-Sep-80      INIT  .OBJ      2P 03-Sep-80
ERROR .OBJ      2P 03-Sep-80      LWAIT .OBJ      1P 03-Sep-80
CVSWG .OBJ      1P 03-Sep-80      KBCD2B.OBJ      1P 03-Sep-80
KB2BCD.OBJ      1P 03-Sep-80      FLT16 .OBJ      1P 03-Sep-80
INT16 .OBJ      1P 03-Sep-80      IADC  .OBJ      1P 03-Sep-80
ISOADC.OBJ      1P 03-Sep-80      RTS   .OBJ     12P 03-Sep-80
ISORTS.OBJ      4P 03-Sep-80      IDAC  .OBJ      1P 03-Sep-80
DISP  .OBJ      2P 03-Sep-80      SETR  .OBJ      4P 03-Sep-80
HIST  .OBJ      7P 03-Sep-80      IDIR  .OBJ      2P 03-Sep-80
IDOR  .OBJ      2P 03-Sep-80      DRS   .OBJ      6P 03-Sep-80
DPOLL .OBJ      2P 03-Sep-80
 31 Files, 131 Blocks
 355 Free blocks
```

APPENDIX B

Sample Throughputs

B.1 Test Conditions

The following throughput tests were all made on Data Translation LAB-DATAX laboratory computer systems. These systems all had the following in common:

1. LSI-11/2 processor (which does not use microcode refresh) equipped with floating point instruction set (DTLIB does not explicitly use this option)

2. KPV-11 power sequencer and program-controlled Line Time Clock

3. 64 Kbytes of memory, of which 56 Kbytes can be addressed by the processor

4. RX02-compatable floppy diskette system (1 Mbyte of storage)

5. RT-11 V4 SJ monitor, as distributed by DEC (no SYSGEN options)

6. FORTRAN/RT-11 V2.1 compiler with all DEC-distributed patches to 8/11/80

7. Appropriate Data Translation analog and digital peripherals with the LAB-DATAX BNC front-panel connections.

8. Laboratory-grade test equipment as needed for signal generation and measurement

B.2 RTS tests

RTS operating under FAST SWEEP mode

Interface product	IPRIO=7	IPRIO=0
DT2782	41.7 KHz	15.2 KHz
DT2762	23.3 KHz	11.1 KHz

Notes:

The above rates represent 4000-word transfers, with RTS being called repeatedly without the error flag becoming set (ICMF=-1). The difference between IPRIO=7 and IPRIO=0 represents the servicing of interrupts not related to the A/D data collection taking time from the processor.

RTS under normal collection (mode 0 or 2)

Interface product	NCCH=1	NCCH=8
DT2782	11.1 KHz	1.58 KHz
DT2762	10.2 KHz	1.14 KHz

Notes:

The above rates represent 4000-word transfers under interrupt driven collection, occurring while both Line Time Clock interrupts were being serviced along with terminal I/O interrupts. These rates were achieved continuously with repeated RTS calls error-free. Higher rates could be achieved in bursts (for example, between 60 Hz LTC interrupts).

RTS operating under DMA

Burst Mode

Interface product	NCCH=1	NCCH=8
DT2782	149.2 KHz	68.5 KHz
DT2762	not applicable	not applicable

Burst Mode Disabled

Interface product	NCCH=1	NCCH=8
DT2782	33.3 Hz	12.8 KHz
DT2762	not applicable	not applicable

Notes:

The A/D converter on this interface is rated at 125 KHz. Higher rates can be achieved on a single channel due to the bypassing of the analog multiplexor inter-channel settling time. Under DMA with burst mode enabled, the entire buffer is sampled in one sweep if NCCH=1. Thus, the 33 Hz time represents how often the entire 4000 word transfer could be

accomplished (represents an effective 133 KHz conversion rate, which is similar to the 149 KHz rate under non-burst, but includes inter-buffer software overhead also, which lowers the overall rate).

B.3 SETR tests

Using a +/- 10V triangle wave as an input to Schmitt Trigger #1

SETR (operating at ST1 rate)

simple compl. routine, mode 1	926 Hz
no completion routine, mode 1	4.17 KHz
low overhead interrupts	4.76 KHz

APPENDIX C

Sample Configuration File

```
; Sample DTLIB configuration file
;
; Remember to precede all comment lines with semi-colons
;
; In addition, all numbers typed will be interpreted as octal
; numbers unless a decimal point is typed immediately after
; the number. This is important to remember when typing the
; number of channels.
;
; Generate start of file
;
        .MCALL  START
        START
;
; define analog inputs
;
; DT2762-PG-SE  at factory standard addresses
; will become analog input unit #0
;
AINPUT BASE=177000,VECTOR=130,NUMCH=16.,PG=PRESENT
;
; DT2782-8DI-C at factory standard addresses
; will become analog input unit #1
;
AINPUT BASE=170400,VECTOR=400,NUMCH=8.,DMA=PRESENT
;
; DT2781-8DI analog I/O, define just the input part here
;
;  user-set addresses (not factory standard)
; will become analog input unit #2
;
AINPUT BASE=177010,VECTOR=140,NUMCH=8.
;
;
; DT2769 real-time clock
;  factory standard addresses
; will become real-time clock unit #0
;
CLOCK  BASE=170420,VECTOR=440
```

```
; DT2766 as analog output unit #0
; bipolar, 12 bits, customer selected addresses
; will become analog output unit #0
; also define another similar unit as unit #1
;
AOUTPUT         BASE=170450,NUMCH=4,TYPE=BIPOLAR
AOUTPUT         BASE=170460,NUMCH=4,TYPE=BIPOLAR
;
; now define output portion of DT2781 mentioned earlier
;
AOUTPUT         BASE=177014,NUMCH=2,TYPE=BIPOLAR
;
; DT2771 point plotter at factory standard addresses
;
PLOTTER         BASE=170440,VECTOR=310
;
;
; define pseudo-digital unit for DRS usage in sampling memory
;
DIGITAL         MEMORY=ACTIVE
;
; define 3 digital I/O units at customer selected addresses
;
DIGITAL         BASE=170470,VECTOR=450
DIGITAL         BASE=170500,VECTOR=460
DIGITAL         BASE=170510,VECTOR=470
;
;
; all done with sample (remember that all numbers will be
; evaluated as octal numbers unless a decimal point follows
; the number to indicate a decimal number).
;
;
        DONE
        .END
```

APPENDIX D

Sample Programs

This appendix contains listings of the FORTRAN example programs distributed with DTLIB. The listings begin on the next page.

D.1 Example Program 1

```
        PROGRAM EXMPL1
C
C       Example #1
C
C Using DTLIB subroutines HIST and SETR to measure the
C frequency of an external frequency source. This example
C has been written assuming the frequency of the external
C signal to be approximately 1 KHz. For demonstration
C purposes a 1 KHz pulse train should be connected to the
C Schmitt Trigger #2 input of the DT2769 Real-Time Clock
C board.
C
C Signal connections:
C
C TTL level, without external level potentiometers connected
C to DT2769:
C       The TTL level select switch for ST2 (SW#3 switch#4)
C       must be turned ON to select the TTL level for ST2
C       firing, and the variable level select switch (SW#3
C       switch#3) must be turned OFF. The firing polarity
C       switch can be in either the ON or OFF position.
C
C non-TTL level or TTL level, but using either on-board
C or off-board level potentiometers:
C       The TTL level select switch for ST2 (SW#3 switch#4)
C       must be turned OFF, and the variable level select
C       switch (SW#3 switch#3) must be turned ON to select
C       variable control of the ST2 firing level. The firing
C       polarity can be in either the ON or OFF position.
C
C Note: SW#3 switch #4 means switch pack #3, switch section #4
C
C With these external connections and adjustments made the
C user can start running this test program which will display
C the frequency of the input signal on the user terminal.
C
C The procedure to measure the frequency will be to set up
C the HIST routine to take 10 readings of the Buffer/Preset
C Register in mode 3 clock operation running at 100 Khz. The
C SETR routine will be used to actually set the clock for
C mode 3 operation at 100 Khz. After the ten BPR readings are
C obtained, the count is converted to a time interval and then
C averaged over the ten readings. The final average time
C interval then represents the period of the input signal from
C which frequency is obtained by inversion.
C
C Above a 3 KHz rate there may be an increasing number of "hits"
C between the HIST routine performing sampling and the external
C signal, causing hardware flag errors on the Real Time Clock.
C In this sample program, this will cause the FREQUENCY TOO HIGH
C message to be displayed.
```

```
C       Declare some required buffer space.
C
        DIMENSION BUF(10),IBUF(10)
C
C       Number of BPR reading : NREAD=10
C
        NREAD=10
C
C       Clock unit number (change to operate on another clock)
C
        IUNIT=0
C
C       Clock operation rate : IRATE=2 (100 KHz)
C
        IRATE=2
C
C       Clock operation mode : MODE=3+4=7
C       where: 3=external event (ST2) from zero base
C              4=start clock on ST2
C
        MODE=7
C
C       Tell user about program, then write freq of 0 message while
C       collecting first buffer of data
C
        WRITE(5,900) IUNIT,NREAD,IRATE,MODE
900     FORMAT(1H ,'Example program to measure the frequency of an'/
     #  1H ,'external signal connected to the ST2-in line'/
     #  1H ,'of clock # ',I2,'. The period will be averaged'/
     #  1H ,'over ',I2,' readings with the clock IRATE = ',I1/
     #  1H ,' and MODE = ',I1/////)
        FREQ=0.0
        GOTO 25
C
C       Set all completion/error flags to zero
C
1       ICMF=0
        IBEF=0
        ICMF2=0
C
C       Set the ISIZ variable to the dimension
C       of the buffer array IBUF : ISIZ=10
C
        ISIZ=10
C
C       Now call the HIST routine to set up the
C       software for the required data.
C
        CALL HIST(IBUF,ISIZ,,NREAD,IUNIT,,,ICMF,IBEF)
C
C       Now start the clock via SETR, the previous
C       call to HIST set up all the HIST parameters
C
        CALL SETR(IRATE,IUNIT,MODE,0,ICMF2)
```

```
C
C       Now wait until the HIST routine completes
C       its data gathering. This will be signaled
C       by the completion/error flag going to 1
C       from the zero state.
C
        CALL LWAIT(ICMF,0)
C
C       Check for frequency too high (ST2 events too fast
C       for software interrupts to handle)
C
        IF (ICMF .EQ. -1) GO TO 30
C
C       Now convert the input time readings of the BPR
C       into real-numbers and then convert the numbers to
C       milliseconds. The conversion of the clock counts
C       to REAL*4 data is accomplished by means of the
C       FLT16 function.
C
        DO 10 I=1,NREAD
        X=FLT16(IBUF(I))
        BUF(I)=X*0.01
10      CONTINUE
C
C       Now average the period over the ten data points.
C
        SUM=0.0
        DO 20 I=1,NREAD
        SUM=SUM+BUF(I)
20      CONTINUE
        PERIOD=SUM/NREAD
C
C       Convert the average period into frequency.
C
        IF (PERIOD.EQ.0) GOTO 1
        FREQ=1.0/PERIOD
C
C       Check for input frequency too low to be displayed
C
        IF (FREQ .LT. 0.01) GO TO 40
C
C       Print out the current value of the frequency
C       on the terminal.
C
25      WRITE(5,1000) FREQ
1000    FORMAT(1H+,'Frequency= ',F4.2,' KHz')
        GO TO 50
```

```
C
C       Messages for limits of frequency response.
C
30      WRITE(5,1001)
1001    FORMAT(1H+,'Frequency too high ')
        GO TO 50
C
40      WRITE (5,1002)
1002    FORMAT(1H+,'Frequency too low  ')
C
50      CONTINUE
C
C       Turn off the clock, so that it can be
C       restarted for the new iteration.
C
        CALL SETR(-1,IUNIT,0,0,0)
C
C       Go back for another iteration.
C       To stop the program, type two CONTROL-Cs,
C       which will exit back to RT-11.
C       (Remember to do a RESET command in RT-11
C       before running another program)
C
        GO TO 1
        STOP
        END
```

D.2 Example Program 2

```
      PROGRAM EXMPL2
C
C     Example #2
C
C     This example illustrates how the SETR subroutine
C     is used to operate the Real-Time clock in an
C     non-interrupt driven mode to act as a programmable
C     pulse generator.
C
C     Declare the clock rate array
C
      DIMENSION RATE(6)
C
C     Assign the 5 clock rates and a sixth stop rate
C
      DATA RATE/1.0E6,1.0E5,1.0E4,1.0E3,1.0E2,0.0/
C
C     preset the clock unit # and tell user (change this
C     line to select other clocks)
C
      IUNIT=0
      WRITE(5,99) IUNIT
99    FORMAT(1H ,'Operating on clock unit ',I1/
     # 1H ,'Enter a frequency of zero to exit.'/)
C
C     Query the user for the required frequency
C
1     WRITE(5,100)
      READ(5,101) FREQ
100   FORMAT(1H$,'Frequency? ')
101   FORMAT(E10.0)
C
C     Exit Check - if FREQ=0, exit
C
      IF (FREQ .EQ. 0.0) GO TO 999
```

```
C
C       Start the IRATE and PRESET finding iteration.
C       This iteration process will start at the fastest
C       oscillator frequency and work its way down to
C       slower frequencies, trying to find the RATE/PRESET
C       combination that will most accurately synthesize
C       the requested frequency. If the frequency can't
C       even be approximated, the program will report
C       that fact to the user.
C
C start at IRATE = 1, array RATE(i) = corresponding base
C       frequency
C
        IRATE=0
2       IRATE=IRATE+1
C
C determine the PRESET value needed to synthesize the
C requested frequency with the main clock oscillator
C set to IRATE.
C
        PRESET=RATE(IRATE)/FREQ
C
C if PRESET is less than 1 and we are at the 1MHz base
C rate, then the requested frequency is too fast.
C
        IF ((PRESET.LT.1.0).AND.(IRATE.EQ.1)) GO TO 3
C
C if PRESET is too large (>65535) and there are still more
C slower base rates to check, loop back and shift clock to
C next slower rate.
C
        IF((PRESET.GT.65535.0).AND.(IRATE.LE.5)) GO TO 2
C
C if we had to step all the way to 6, the requested frequency
C is too slow - report.
C
        IF(IRATE.EQ.6) GO TO 4
C
C for later use, make PRESET rounded to the nearest integer value
C while keeping it a REAL*4 value
C
        PRESET=AINT(PRESET)
```

```
C
C       Here we have a PRESET and IRATE in range, so
C       we first call SETR to turn off the clock
C       from the last set value, then we will call
C       SETR to set the clock running at the new
C       rate and preset. The clock must be turned
C       off before reseting it to avoid a device
C       conflict error (in case a previous SETR call
C       was still active on the clock). When SETR is
C       used to set the clock running at the new rate
C       the mode selected is 1+8=9 where
C               1 selects repeated interval
C               8 selects no interrupts
C                       (we only need to generate hardware
C                       signals, not software interrupts)
C
        CALL SETR(-1,IUNIT,,,)
        CALL SETR(IRATE,IUNIT,9,PRESET,ICMF)
C
C       After setting the clock for the new rate
C       we report the actual frequency synthesized to the
C       user, then return and query the user again.
C
        WRITE   (5,170) FREQ,RATE(IRATE)/PRESET
        GO TO 1
C
170     FORMAT  (1H ,'Requested frequency: ',G12.6,'  Synthesized',
     #  ' frequency: ',G12.6)
C
C       The various error reporting code follows each
C       of which returns to the query point after the
C       error is reported.
C
3       WRITE(5,150)
150     FORMAT(1H ,'Requested frequency is too fast to synthesize'/)
        GO TO 1
C
4       WRITE(5,160)
160     FORMAT(1H ,'Requested frequency is too slow to synthesize'/)
        GO TO 1
C
C insure that clock has stopped before exiting (good programming
 practice)
C
999     CALL SETR(-1,IUNIT,,,)
        STOP
        END
```

D.3 Example Program 3

```
      PROGRAM EXMPL3
C
C     Example #3
C
C     This sample program illustrates the usage of the
C     IADC subroutine for gathering A/D data. The program
C     will input data from a user specified channel and
C     display the reading as a voltage on the user terminal
C     in units of volts.
C
C Variable declarations
C
      INTEGER ICHAN,IUNIT,ISTAT,INPUT
      REAL    FSR,LSBVAL,VOLTS
C
C Define the value of the Least Significant Bit from the A/D
C
C Note that the A/D data returned is already in FORTRAN integer
C form. You can display the number in octal, decimal, or whatever,
C but the number itself represents the voltage as the number of
C LSB pieces needed to duplicate the voltage. Therefore, to convert
C this number into a number that represents standard engineering
C units (volts, millivolts, etc) a numeric conversion is necessary.
C In this case, the A/D binary data will be converted to volts.
C
C     Declare the LSB weight for a +/- 10.0 volt system
C
C FSR is the A/D converter's Full Scale Range. For a +/- 10 volt
C converter, this means that its FSR is 20 volts. Change this
C program line if a different FSR applies to your A/D.
C
      FSR = 20.
C
C with the Full Scale Range defined, the value of the Least Significant
C bit is computed by dividing the FSR by the resolution of the A/D
C converter. For a 12 bit converter, this is 2^^12 or 4096. Change
C this program line if a 14 or 16 bit A/D converter is being used.
C
      LSBVAL = FSR / 4096.
C
C     This line sets up the Job Status Word so that the ITTINR
C     subroutine will work under the FB monitor.  For more
C     information, see the RT-11 Programmer's Reference, under
C     ITTINR.
C
      CALL IPOKE("44,"100.OR.IPEEK("44))
C
C     Start out by querying the user for the unit and channel.
C
      WRITE(5,97)
97    FORMAT(1H ,'Entering -1 for the channel allows the'/
     #  1H ,'unit to be reset.  Use control-C to exit.'/)
```

```
C
5       WRITE(5,98)
        READ(5,99) IUNIT
98      FORMAT(1H$,'Unit? ')
99      FORMAT(I2)
C
10      WRITE(5,100)
        READ(5,101) ICHAN
100     FORMAT(1H$,'Channel? ')
101     FORMAT(I2)
C
C Unit number reset check
C
        IF (ICHAN .EQ. -1) GO TO 5
C
        WRITE(5,102)
102     FORMAT(1H ,'Type a carriage return to modify the channel.'/)
C
C       For this example we will fix the gain to 1
C       which is accomplished by defaulting the gain
C       parameter in the IADC call. In this example
C       IADC is being used as a function, one could
C       just as easily use it as a subroutine by calling
C       it e.g. CALL IADC(ICHAN,IUNIT,,INPUT)
C       in the above call statement note that the gain
C       parameter has been defaulted.
C
20      INPUT=IADC(ICHAN,IUNIT)
C
C       Convert the input to voltage by using the LSB
C       conversion weighting.
C
        VOLTS=LSBVAL*INPUT
C
C       Print out the current voltage to reading to
C       the user terminal.
C
        WRITE(5,120) VOLTS
120     FORMAT(1H+,'VOLTAGE= ',F7.3)
C
C       Now check the TTY to see if the user wants to
C       change the channel (signified by typing a
C       return) or not.
C
        ISTAT=ITTINR()
        IF(ISTAT.LT.0) GO TO 20
        GO TO 10
        STOP
        END
```

D.4 Example Program 4

```
      PROGRAM EXMPL4
C
C     Example #4
C
C This example is to illustrate the use of SETR and RTS in combination
C to perform real-time clock-driven sampling. Although the CLOCK
C OVERFLOW signal is used in this example, any TTL trigger signal
C (subject to the the requirements of the A/D board, see hardware
C manual) could be used. If the clock is not used, remove the SETR
C calls.
C
C In this example the SETR routine will first be called to set the
C DT2769 Real-Time clock running as a free-running pulse generator at a
C 1 Hz frequency so that the clock overflow pulses that will be used to
C trigger the A/D system will trigger a conversion every 1 second.
C After the clock has been set up as a hardware trigger source, the RTS
C routine will then be called an instructed on how to collect A/D data.
C Note that the RTS always requires some external hardware trigger
C signal to actually trigger the A/D converter - in this example, the
C Real Time Clock is being used to generate this hardware signal. The
C user must insure that the CLOCK OVERFLOW signal from the clock is
C connected to the RTC INL signal of the A/D converter (a special small
C wire is shipped with all DTI real time clocks for this purpose).
C
C The RTS routine will be used to collect 10 samples in the input
C buffer. Since the clock is generating trigger signals every second,
C this collection process will take 10 seconds. After the buffer is
C filled the data obtained is displayed and a new sampling is done for
C ten more points.
C
C It is important to remember that under nearly all operating modes,
C the RTS routine collects data asynchronously. That is, the RTS
C routine will return back to the main-line FORTRAN program immediately
C after setting up the sampling. While the FORTRAN program continues to
C execute, DTLIB is collecting the data using the PDP-11 interrupt
C structure from the A/D converter. This means that the data will
C gradually fill the input buffer as data is collected, but that the
C data buffer initially is empty (contains random data). In order for
C the FORTRAN program to determine when all of the data is ready and
C that DTLIB has finished collecting data, the main program must
C monitor the flag variable ICMF. When ICMF changes from the initial 0
C to a 1, this means that DTLIB has finished collecting all the data
C and the data buffer is now valid. If ICMF changes from the initial 0
C to a -1, this means that DTLIB encountered some type of sampling
C error while taking data, and that some or all of the data buffer may
C not be valid.
```

```
C
C Declare the buffer and other variables
C
      DIMENSION IBUF(10)
      INTEGER IBUF,ICMF,ICMF1,MODE,MODE1,IUNIT,IUNIT1
      INTEGER ISIZ,NBUF,NREAD,ISTCHN,NCCH,IGAIN,IBEF
      REAL    PRESET
C
C talk to user
C
      WRITE(5,100)
100   FORMAT(1H ,'Sample program for real-time A/D operation'//)
C
C     Start the clock for free running operation at a 1 Hz rate
C     with no software interrupts (hardware overflow only)
C     so MODE=1+8=9, PRESET=100. , IRATE=5
C
      MODE1=9
      PRESET=100.
      IRATE=5
      IUNIT1=0
      ICMF1=0
      CALL SETR(IRATE,IUNIT1,MODE1,PRESET,ICMF1)
C
C     Now call RTS to perform the real time sampling
C     for ten samples. Before RTS is entered the completion
C     error flag must always be reset.
C
C The parameters used here are:
C
C           ISIZ    = size of buffer        = 10
C           NBUF    = #of subbuffers        = 1
C           NREAD   = #of data points       = 10
C           ISTCHN  = start channel         = 0
C           NCCH    = consecutive chan      = 1
C           IGAIN   = amplifier gain        = 1
C           MODE    = CLK OVF driven        = 2
C
      ISIZ=10
      NBUF=1
      NREAD=10
      ISTCHN=0
      NCCH=1
      IGAIN=1
      MODE=2
1     ICMF=0
      IUNIT=0
C
C make the call to RTS
C
      CALL RTS(IBUF,ISIZ,NBUF,NREAD,ISTCHN,NCCH,IUNIT,IGAIN,MODE,
     #  ICMF,IBEF)
```

```
C
C       Print out the waiting for data message
C
        WRITE(5,110)
110     FORMAT(1H ,'Data acquisition in progress ....'/)
C
C       Wait for the completion flag to be set
C
        CALL    LWAIT(ICMF,0)
C
C       The completion flag is set so data acquisition
C       is complete, now print out the message that
C       the data has been gathered and then print out the
C       actual ten data values.
C
        IF (ICMF.EQ.-1) GOTO 130
C
        WRITE(5,115)
C
C Print the data. Note that the data is already in FORTRAN INTEGER
C form, so that any desired formatting may be used for output. In
C this case, the data is being displayed in octal.
C
        WRITE(5,120) (IBUF(I),I=1,10)
115     FORMAT(1H ,'Data acquisition complete'/)
120     FORMAT(1H ,'A/D= ',O6)
        GO TO 1
C
C error reporter
C
130     WRITE (5,140)
140     FORMAT (1H ,'Collection error'/)
        GOTO 1
        END
```

D.5 Example Program 5

```
        PROGRAM EXMPL5
C
C*************************************************************************
C*                                                                       *
C*      Example program to demonstrate use of completion routines in     *
C*      conjunction with file data storage.                              *
C*                                                                       *
C*************************************************************************
C
C This program utilizes the RTS subroutine to collect large numbers
C of data points and store them on disk in an output file.  The number
C of points and the file name can both be specified by the user.
C
C The data collection utilizes the sub-buffer feature of RTS;  the main
C buffer is 1000 points, split into ten 100-point sub-buffers.  RTS
C fills these buffers as if they were arranged in a ring, that is, it
C starts with sub-buffer number 0, then number 1, and so on up to
C number 9, then back to number 0 again. It continues looping around in
C this way until the desired number of data points have been collected.
C The variable IBEF indicates the number of buffers which are not
C filled with data at any given time. At the start of program
 execution,
C this number should be 10; each time that RTS fills a buffer, IBEF
 will
C be decremented by one.
C
C The subroutine SAVEM takes the data from the buffer and writes it out
C to the user's file, one sub-buffer at a time; SAVEM is called by RTS
C every time that a sub-buffer is filled.
C
C The subroutine SAVEM is in a separate source file, and must be
C compiled separatly. When linking, include both this object file
C and the SAVEM object file in the list of modules to be linked.
C
C Note:
C  Under the RT-11 operating system, this program should be compiled
C  with the /NOSWAP option; this will insure proper operation of the
C  system subroutine GETSTR.
C
C*************************************************************************
```

```
C*************************************************************************
C
C       Program declarations.
C
        EXTERNAL SAVEM
        COMMON/SAVEM/IBEF,NBUF,NCNT,NSUBSZ,NLINE,
     #               FORMAR(20),SCALE,NOUT,NREAD,IBUF(1000)
        LOGICAL*1 INPSTR(15)     !Used to input the user's filename
        LOGICAL*1 ERRFLG         !An error flag set by GETSTR
C
C       Initialize the program parameters.
C
C       The format of the output line: (in 4 char chunks, REAL*4
C       holds 4 char)
C       (1H ,9(F7.3,1X),F7.3)
C
        DATA FORMAR(1),FORMAR(2),FORMAR(3),FORMAR(4),FORMAR(5),
     # FORMAR(6)
     #       /'(1H ',',9(F','7.3,',','1X),',','F7.3',')   '/
C
        NBUF = 10                !The number of sub-buffers
        NCNT = 0                 !The number of the next sub-buffer
        ISIZ = 1000              !Size of the buffer
        NSUBSZ = ISIZ/NBUF       !The size of each sub-buffer
        NLINE = 10               !The number of data points printed
C                                !on each line of the output file
        SCALE = 20./4096         !Scale factor for +/- 10V inputs
        NOUT = 0                 !The number of points in the  file
```

```
C*************************************************************************
C
C       Open files and query user for number of points to collect
C
C
C       Get the output file name from the user and open it
C
10      WRITE(5,900)
900     FORMAT(1H$,'Enter the name of the output file - ')
        CALL GETSTR(5,INPSTR,14,ERRFLG)
        IF (ERRFLG) GO TO 10
        OPEN(UNIT=2,NAME=INPSTR,TYPE='NEW')
C
C Write header to file (will also set up all RT-11 file operations)
C
        WRITE(2,905,ERR=12,END=12)
        GOTO 15
905     FORMAT(1H ,'Data file written by DTLIB sample program EXMPL5')
C
C       Some type of file error seen - report and try again
C
12      WRITE(5,907)
        GOTO 10
907     FORMAT(1H ,'Error in opening/starting data file')
C
C       Get the desired number of data points from the user.
C
15      WRITE(5,910)
910     FORMAT(1H ,'Enter the number of data points desired'/
     #  1H$,'(>100 points, multiple of 100 points) -- ')
        READ(5,920) NREAD
920     FORMAT(I5)
C
        IF (NREAD .LE. 100) GO TO 15    !If invalid number of points
```

Example Program 5

```
C*************************************************************************
C
C       Set the real-time clock for 100 Hz rate.
C
        IRATE = 5           !Set for 100 Hz count rate
        IUNIT1 = 0          !Set for clock unit #0
        MODE = 9            !Repeated interval mode; disable the
C                           !software clock
        PRESET = 1.         !Set to output pulse every count
        ICMF1 = 0           !Initial completion flag
C
        CALL SETR(IRATE,IUNIT1,MODE,PRESET,ICMF1)
C
C For efficiency reasons, the number of points sampled by RTS must be
C a multiple of NSUBSZ.  So the first step is to compute a new argument
C IREAD, which is NREAD rounded up to the nearest multiple of NSUBSZ.
C
        IREAD = (NREAD / NSUBSZ) * NSUBSZ       !Rounded down value
        IF (NREAD .GT. IREAD) IREAD = IREAD + NSUBSZ
C
C Setup the real-time sampling routine for a ten-ring buffer,
C to sample IREAD points, NSUBSZ at a time, and store them away using
C the completion routine SAVEM.
C
        ISTCHN = 0          !Starting channel of A/D input device
        NCCH = 1            !Number of channels to sample
        IUNIT2 = 0          !Unit number of A/D input
        IGAIN = 1           !Gain setting of A/D input
        MODE = 2            !Set mode to use real time clock input trigger
        ICMF = 0            !Initialize completion flag
        IBEF = 0            !Initial buffer event flag; will be set to NBUF
C
        CALL RTS(IBUF,ISIZ,NBUF,IREAD,ISTCHN,NCCH,
     #           IUNIT2,IGAIN,MODE,ICMF,IBEF,,SAVEM)
```

```
C*************************************************************************
C
C       Wait for the data to be collected.
C
        WRITE(5,940)
940     FORMAT(1H ,'Data acquisition in progress ....'/)
        CALL    LWAIT(ICMF,0)
C
C       If successful collection, indicate so to the user.
C
        IF (ICMF .EQ. -1) GO TO 20
        WRITE(5,950)
950     FORMAT(1H ,'Data acquisition complete'/)
C
C       Close the output file.
C
        CLOSE(UNIT=2)
        STOP
C
C       Terminate here if an error occurs.
C
20      WRITE(5,120)
120     FORMAT(1H ,'Error in data acquisition'/)
C
C       Close the output file anyways, in case something was written
C
        CLOSE(UNIT=2)
        STOP
        END
```

D.5.1 Completion routine for Example 5

```
      SUBROUTINE SAVEM
C*************************************************************************
C
C This routine is designed to write data generated by the RTS
C A/D sampling routine to an output file.  It is intended to be
C called using the completion routine mechanism, and receives its
C arguments through the common block SAVEM:
C
      COMMON/SAVEM/IBEF,                !DTLIB Buffer Event Flag
     #             NBUF,                !Number of sub-buffers
     #             NCNT,                !Current output sub-buffer
     #             NSUBSZ,              !Size of each sub-buffer
     #             NLINE,               !Number of data points per line
     #             FORMAR(20),          !The format array
     #             SCALE,               !Scale factor for data points
     #             NOUT,      !Number of data points already output
     #             NREAD,     !Total number of points to be output
     #             IBUF(1)    !The buffer (Actual dimension unknown)
C
C       There are some initialization requirements:
C
C NBUF and NREAD should be initialized to appropriate values as
C detailed in the DTLIB documentation; note that these values should
C not change between calls to SAVEM.
C
C  NCNT and NOUT should be initialized to 0.
C  NSUBSZ should be initialized to the sub-buffer size.
C  NLINE and FORMAR should be set to the desired output format.
C  SCALE should be set to the desired scale factor; this is usually
C  the voltage range divided by 2^N, where N is the number
C  of bits of data.
C
C SAVEM considers NCNT and NOUT to be internal variables; they only
C appear in the common block for the purpose of correct initialization.
C
C The following buffer is used to convert the input data from its
C integer form to the real output form.  Note that because it is
C limited in size to 20, that NLINE should never exceed 20.
C
      DIMENSION BUF(20)
C
C*************************************************************************
```

```
C*******************************************************************************
C
C       Compute the sub-buffer limit addresses.
C
        INITAD = NSUBSZ * NCNT
        IFINAD = INITAD + NSUBSZ - 1
C
C       Within double loop, convert the integer data to real,
C       then transmit it to the output file one line at a time.
C
                DO 20 I = INITAD, IFINAD, NLINE
                NLIMIT = MIN0(NLINE,(NREAD-NOUT))
                IF (NLIMIT .EQ. 0) RETURN
C
                        DO 10 K = 1, NLIMIT
                        BUF(K) = IBUF(I + K) * SCALE
10                      CONTINUE
C
                WRITE(2,FORMAR,ERR=800,END=810) (BUF(J), J=1,NLIMIT)
                NOUT = NOUT + NLIMIT
20              CONTINUE
C
        NCNT = NCNT + 1
        IBEF = IBEF + 1
        IF (NCNT .LT. NBUF) RETURN
        NCNT = 0
        RETURN
C
C       Error return for bad file operation
C
800     WRITE(5,910)
910     FORMAT(1H ,'Error during write to data file'/)
        RETURN
C
C       End-of-file encountered
C
810     WRITE(5,920)
920     FORMAT(1H ,'End-of-file encountered during write to file'/)
        RETURN
        END
```

APPENDIX E

DTLIB End User Binary License Agreement

This License Agreement, made in Massachusetts, is effective from the date on which it is accepted by Data Translation Incorporated, a Massachusetts Corporation (hereafter "DTI"). DTI agrees to license

_____
Name

_____
Company or University

_____
Address

_____

_____

(hereafter "CUSTOMER") to operate DTLIB* (hereafter "SOFTWARE"), a real-time software package supporting analog and digital peripherals under FORTRAN IV, on a single Central Processing Unit as described by

_____
Central Processor Serial Number and Model

_____
Address

_____

(hereafter "CPU"), subject to the following terms:

= = = = = = = = = = = = =

* DTLIB is a trademark of Data Translation Incorporated.

1. This License is non-exclusive and non-transferable.

2. SOFTWARE is furnished for use only on specified CPU and may only be copied, in whole or in part (with the proper inclusion of copyright notic on SOFTWARE), for use on such CPU.

3. SOFTWARE is furnished to operate in conjunction with the Digital Equipment Corporation RT-11* operating system (SJ or FB, Version 3B or greater) executing on an LSI-11* or LSI-11/2* processor.

4. CUSTOMER warrants that he is licensed by Digital Equipment Corporation to operate the selected operating system on specified CPU.

5. SOFTWARE is furnished to operate in conjunction with the Digital Equipment Corporation FORTRAN/RT-11 language compiler, Version 2.1 or greater.

6. CUSTOMER warrants that he is licensed by Digital Equipment Corporation to operate the selected language compiler on specified CPU.

7. CUSTOMER shall not provide or otherwise make available SOFTWARE or any portion thereof to any third party without the prior written approval of DTI.

8. Title to and ownership of SOFTWARE shall at all times remain with DTI.

9. CUSTOMER shall pay to DTI a fixed fee of seven hundred ninety-five dollars ($795.00) payable upon receipt of DTI invoice.

10. CUSTOMER shall be responsible for all taxes arising from this agreement, except for DTI income taxes.

11. DTI shall not be liable for any losses or damages resulting from CUSTOMER'S use of SOFTWARE. IN NO CASE SHALL DTI BE LIABLE FOR CONSEQUENTIAL DAMAGES.

12. DTI warrants that SOFTWARE operates as represented and is free from defects. For a period of one year from the date of execution of this agreement, all defects encountered by CUSTOMER will be repaired without charge to CUSTOMER. NO OTHER WARRANTY IS EXPRESSED OR IMPLIED.

13. SOFTWARE and any subsequent modifications to SOFTWARE will be distributed as binary files in RT-11 compatible form on magnetic media purchased by DTI and billed to CUSTOMER.

= = = = = = = = = = = = =

* RT-11,LSI-11, and LSI-11/2 are trademarks of Digital Equipment Corporation.

The type of medium to be used is _____.

14. CUSTOMER shall provide DTI with the name(s) of CUSTOMER personnel to contact concerning SOFTWARE should assistance be required in using or repairing SOFTWARE.

_____
Authorized Software Contact(s)

_____
Telephone,TELEX, or TWX Number of Contact(s)

15. Each person executing this agreement on behalf of a Corporation, Partnership, or Limited Partnership, as the case may be, warrants that he is duly authorized by said entity to execute this agreement.

16. Violation by CUSTOMER of any of the above terms will result in revocation of the license granted by this agreement. Upon revocation of license, all copies of SOFTWARE electronic, magnetic, or photographic and any facsimiles thereof in possession of CUSTOMER must be surrendered to DTI.

	ACCEPTED
Responsible Signer, CUSTOMER	Data Translation Incorporated
_____	_____
Name	Name
_____	_____
Title	Title
_____	_____
Date	Date
	Logged, DTI Software Support
	_____
	Binary License Number
	_____
	Date

INDEX

DATA TRANSLATION
INC
4 Strathmore Rd., Natick MA 01760
(617) 655-5300 Telex 948474